HOL-CSP_RS: CSP Semantics over Restriction Spaces

Benoît Ballenghien Burkhart Wolff

June 3, 2025

Abstract

We use the `Restriction_Spaces` library as a semantic foundation for the process algebra framework `HOL-CSP`, offering a complementary backend to the existing `HOLCF` infrastructure. The type of processes is instantiated as a restriction space, and we prove that it is complete in this setting. This enables the construction of fixed points for recursive process definitions without having to rely exclusively on a pointed complete partial order. Notably, some operators are constructive without being Scott-continuous, and vice versa, illustrating the genuine complementarity between the two approaches. We also show that key CSP operators are either constructive or non-destructive, and verify the admissibility of several predicates, thereby supporting automated reasoning over recursive specifications.

Contents

1 Depth Operator **1**
- 1.1 Definition . . . 1
- 1.2 Projections . . . 3
- 1.3 Proof obligation . . . 6
- 1.4 Compatibility with Refinements . . . 9
- 1.5 First Laws . . . 11
- 1.6 Monotony . . . 12
  - 1.6.1 $P \downarrow n$ is an Approximation of the P . . . 12
  - 1.6.2 Monotony of $(\downarrow)$ . . . 13
  - 1.6.3 Interpretations of Refinements . . . 14
- 1.7 Continuity . . . 16
- 1.8 Completeness . . . 17

2 Constructiveness of Prefixes **19**
- 2.1 Equality . . . 19
- 2.2 Constructiveness . . . 21

3 Non Destructiveness of Choices **22**
- 3.1 Equality . . . 22
- 3.2 Non Destructiveness . . . 23

4 Non Destructiveness of Renaming **24**
4.1 Equality 24
4.2 Non Destructiveness 25

5 Non Destructiveness of Sequential Composition **25**
5.1 Refinement 25
5.2 Non Destructiveness 27

6 Non Destructiveness of Synchronization Product **31**
6.1 Preliminaries 31
6.2 Refinement 37
6.3 Non Destructiveness 40

7 Non Destructiveness of Throw **44**
7.1 Equality 44
7.2 Refinement 46
7.3 Non Destructiveness 49

8 Non Destructiveness of Interrupt **56**
8.1 Refinement 56
8.2 Non Destructiveness 57

9 Non too Destructiveness of After **63**
9.1 Equality 63
9.2 Non too Destructiveness 64

10 Destructiveness of Hiding **66**
10.1 Refinement 66
10.2 Destructiveness 68

11 Admissibility **68**
11.1 Belonging 68
11.2 Refining 69
11.2.1 Transitions 70

12 Higher-Order Rules **73**
12.1 Prefixes 73
12.2 Choices 74
12.3 Renaming 75
12.4 Sequential Composition 76
12.5 Synchronization Product 76
12.6 Throw 77
12.7 Interrupt 77
12.8 After 77
12.9 Illustration 78

1 Depth Operator

1.1 Definition

instantiation $process_{ptick}$ :: (*type*, *type*) *order-restriction-space*
begin

lift-definition *restriction-process*$_{ptick}$ ::
‹[('*a*, '*r*) *process*$_{ptick}$, *nat*] ⇒ ('*a*, '*r*) *process*$_{ptick}$›
is ‹λ*P n*. ($\mathcal{F}$ *P* ∪ {(*t* @ *u*, *X*) |*t u X*. *t* ∈ $\mathcal{T}$ *P* ∧ *length t* = *n* ∧ *tF t* ∧ *ftF u*},
$\mathcal{D}$ *P* ∪ { *t* @ *u* |*t u*. *t* ∈ $\mathcal{T}$ *P* ∧ *length t* = *n* ∧ *tF t* ∧ *ftF u*})›
proof −
show ‹*?thesis P n*› (**is** ‹*is-process* (*?f*, *?d*)›) **for** *P* **and** *n*
proof (*unfold is-process-def FAILURES-def fst-conv DIVERGENCES-def snd-conv*, *intro conjI impI allI*)
show ‹([], {}) ∈ *?f*› **by** (*simp add*: *process-charn*)
next
show ‹(*t*, *X*) ∈ *?f* ⟹ *ftF t*› **for** *t X*
by *simp* (*meson front-tickFree-append is-processT*)
next
fix *t u*
assume ‹(*t* @ *u*, {}) ∈ *?f*›
then consider ‹(*t* @ *u*, {}) ∈ $\mathcal{F}$ *P*›
| *t' u'* **where** ‹*t* @ *u* = *t'* @ *u'*› ‹*t'* ∈ $\mathcal{T}$ *P*› ‹*length t'* = *n*› ‹*tF t'*› ‹*ftF u'*› **by** *blast*
thus ‹(*t*, {}) ∈ *?f*›
proof *cases*
assume ‹(*t* @ *u*, {}) ∈ $\mathcal{F}$ *P*›
with *is-processT3* **have** ‹(*t*, {}) ∈ $\mathcal{F}$ *P*› **by** *auto*
thus ‹(*t*, {}) ∈ *?f*› **by** *fast*
next
fix *t' u'* **assume** ∗ : ‹*t* @ *u* = *t'* @ *u'*› ‹*t'* ∈ $\mathcal{T}$ *P*› ‹*length t'* = *n*› ‹*tF t'*› ‹*ftF u'*›
show ‹(*t*, {}) ∈ *?f*›
proof (*cases* ‹*t* ≤ *t'*›)
assume ‹*t* ≤ *t'*›
with ∗(*2*) *is-processT3-TR* **have** ‹*t* ∈ $\mathcal{T}$ *P*› **by** *auto*
thus ‹(*t*, {}) ∈ *?f*› **by** (*simp add*: *T-F*)
next
assume ‹¬ *t* ≤ *t'*›
with ∗(*1*) **have** ‹*t* = *t'* @ *take* (*length t* − *length t'*) *u'*›
by (*metis* (*no-types*, *lifting*) *Prefix-Order.prefixI append-Nil2 diff-is-0-eq nle-le take-all take-append take-eq-Nil*)
with ∗(*2*, *3*, *4*, *5*) **show** ‹(*t*, {}) ∈ *?f*›
by *simp* (*metis append-take-drop-id front-tickFree-dw-closed*)
qed
qed
next

show ‹$(t,\ Y) \in ?f \wedge X \subseteq Y \Longrightarrow (t,\ X) \in ?f$› **for** $t\ X\ Y$
by *simp* (*meson is-processT4*)
next
show ‹$(t,\ X) \in ?f \wedge (\forall\, c.\ c \in Y \longrightarrow (t\ @\ [c],\ \{\}) \notin ?f) \Longrightarrow (t,\ X \cup Y) \in ?f$› **for** $t\ X\ Y$
by (*auto simp add*: *is-processT5*)
next
show ‹$(t\ @\ [\checkmark(r)],\ \{\}) \in ?f \Longrightarrow (t,\ X - \{\checkmark(r)\}) \in ?f$› **for** $t\ r\ X$
by (*simp*, *elim disjE exE*, *solves* ‹*simp add*: *is-processT6*›)
(*metis append-assoc butlast-snoc front-tickFree-dw-closed nonTickFree-n-frontTickFree non-tickFree-tick tickFree-append-iff*)
next
from *front-tickFree-append is-processT7 tickFree-append-iff*
show ‹$t \in ?d \wedge tF\ t \wedge ftF\ u \Longrightarrow t\ @\ u \in ?d$› **for** $t\ u$ **by** *fastforce*
next
from *D-F* **show** ‹$t \in ?d \Longrightarrow (t,\ X) \in ?f$› **for** $t\ X$ **by** *blast*
next
show ‹$t\ @\ [\checkmark(r)] \in ?d \Longrightarrow t \in ?d$› **for** $t\ r$
by *simp* (*metis butlast-append butlast-snoc front-tickFree-iff-tickFree-butlast is-processT9 non-tickFree-tick tickFree-Nil tickFree-append-iff tickFree-imp-front-tickFree*)
qed
qed

1.2 Projections

context fixes P :: ‹$('a,\ 'r)\ process_{ptick}$› **begin**

lemma *F-restriction-process*$_{ptick}$:
‹$\mathcal{F}\ (P \downarrow n) = \mathcal{F}\ P \cup \{(t\ @\ u,\ X)\ |t\ u\ X.\ t \in \mathcal{T}\ P \wedge length\ t = n \wedge tF\ t \wedge ftF\ u\}$›
by (*simp add*: *Failures-def FAILURES-def restriction-process*$_{ptick}$*.rep-eq*)

lemma *D-restriction-process*$_{ptick}$:
‹$\mathcal{D}\ (P \downarrow n) = \mathcal{D}\ P \cup \{t\ @\ u\ |t\ u.\ t \in \mathcal{T}\ P \wedge length\ t = n \wedge tF\ t \wedge ftF\ u\}$›
by (*simp add*: *Divergences-def DIVERGENCES-def restriction-process*$_{ptick}$*.rep-eq*)

lemma *T-restriction-process*$_{ptick}$:
‹$\mathcal{T}\ (P \downarrow n) = \mathcal{T}\ P \cup \{t\ @\ u\ |t\ u.\ t \in \mathcal{T}\ P \wedge length\ t = n \wedge tF\ t \wedge ftF\ u\}$›
using *F-restriction-process*$_{ptick}$ **by** (*auto simp add*: *Failures-def Traces-def TRACES-def*)

lemmas *restriction-process*$_{ptick}$*-projs* = *F-restriction-process*$_{ptick}$ *D-restriction-process*$_{ptick}$ *T-restriction-process*$_{ptick}$

lemma *D-restriction-process$_{ptick}$E*:
assumes ‹$t \in \mathcal{D}\ (P \downarrow n)$›
obtains ‹$t \in \mathcal{D}\ P$› **and** ‹*length* $t \leq n$›
| $u\ v$ **where** ‹$t = u\ @\ v$› ‹$u \in \mathcal{T}\ P$› ‹*length* $u = n$› ‹*tF* u› ‹*ftF* v›
proof −
note *assms* = *that*
from ‹$t \in \mathcal{D}\ (P \downarrow n)$› **have** ‹*ftF* t› **by** (*simp add*: *D-imp-front-tickFree*)
from ‹$t \in \mathcal{D}\ (P \downarrow n)$› **consider** ‹$t \in \mathcal{D}\ P$›
| $u\ v$ **where** ‹$t = u\ @\ v$› ‹$u \in \mathcal{T}\ P$› ‹*length* $u = n$› ‹*tF* u› ‹*ftF* v›
by (*simp add*: *D-restriction-process$_{ptick}$*) *blast*
thus *thesis*
proof *cases*
show ‹$t = u\ @\ v \Longrightarrow u \in \mathcal{T}\ P \Longrightarrow$ *length* $u = n \Longrightarrow$ *tF* $u \Longrightarrow$ *ftF* $v \Longrightarrow$ *thesis*› **for** $u\ v$
by (*fact assms*(*2*))
next
show *thesis* **if** ‹$t \in \mathcal{D}\ P$›
proof (*cases* ‹*length* $t \leq n$›)
from ‹$t \in \mathcal{D}\ P$› **show** ‹*length* $t \leq n \Longrightarrow$ *thesis*› **by** (*rule assms*(*1*))
next
show *thesis* **if** ‹¬ *length* $t \leq n$›
proof (*intro assms*(*2*) *exI*)
show ‹$t =$ *take* $n\ t\ @$ *drop* $n\ t$› **by** *simp*
next
show ‹*take* $n\ t \in \mathcal{T}\ P$› **by** (*metis D-T* ‹$t \in \mathcal{D}\ P$› *append-take-drop-id is-processT3-TR-append*)
next
show ‹*length* (*take* $n\ t$) $= n$› **by** (*simp add*: *min-def* ‹¬ *length* $t \leq n$›)
next
show ‹*tF* (*take* $n\ t$)› **by** (*metis* ‹*ftF* t› *append-take-drop-id drop-eq-Nil2*
front-tickFree-append-iff ‹¬ *length* $t \leq n$›)
next
show ‹*ftF* (*drop* $n\ t$)› **by** (*metis* ‹*ftF* t› *append-take-drop-id drop-eq-Nil*
front-tickFree-append-iff that)
qed
qed
qed
qed

lemma *F-restriction-process$_{ptick}$E* :
assumes ‹$(t,\ X) \in \mathcal{F}\ (P \downarrow n)$›
obtains ‹$(t,\ X) \in \mathcal{F}\ P$› **and** ‹*length* $t \leq n$›
| $u\ v$ **where** ‹$t = u\ @\ v$› ‹$u \in \mathcal{T}\ P$› ‹*length* $u = n$› ‹*tF* u› ‹*ftF* v›
proof −
from ‹$(t,\ X) \in \mathcal{F}\ (P \downarrow n)$› **consider** ‹$(t,\ X) \in \mathcal{F}\ P$› | ‹$t \in \mathcal{D}\ (P \downarrow$

$n)\rangle$
unfolding *restriction-process$_{ptick}$-projs* **by** *blast*
thus *thesis*
proof *cases*
show $\langle(t,\ X) \in \mathcal{F}\ P \Longrightarrow$ *thesis*$\rangle$
by (*metis F-T F-imp-front-tickFree append-take-drop-id drop-eq-Nil front-tickFree-nonempty-append-imp is-processT3-TR-append length-take min-def that*)
next
show $\langle t \in \mathcal{D}\ (P \downarrow n) \Longrightarrow$ *thesis*$\rangle$ **by** (*meson D-restriction-process$_{ptick}$E is-processT8 that*)
qed
qed

lemma *T-restriction-process$_{ptick}$E* :
$\langle[\![t \in \mathcal{T}\ (P \downarrow n);\ t \in \mathcal{T}\ P \Longrightarrow$ *length* $t \leq n \Longrightarrow$ *thesis*;
$\bigwedge u\ v.\ t = u\ @\ v \Longrightarrow u \in \mathcal{T}\ P \Longrightarrow$ *length* $u = n \Longrightarrow$ *tF* $u \Longrightarrow$ *ftF* $v \Longrightarrow$ *thesis*$]\!] \Longrightarrow$ *thesis*$\rangle$
by (*fold T-F-spec, elim F-restriction-process$_{ptick}$E*) (*simp-all add*: *T-F*)

lemmas *restriction-process$_{ptick}$-elims* =
F-restriction-process$_{ptick}$E D-restriction-process$_{ptick}$E T-restriction-process$_{ptick}$E

lemma *D-restriction-process$_{ptick}$I* :
$\langle t \in \mathcal{D}\ P \vee t \in \mathcal{T}\ P \wedge$ (*length* $t = n \wedge$ *tF* $t \vee n <$ *length* $t) \Longrightarrow t \in \mathcal{D}\ (P \downarrow n)\rangle$
by (*simp add*: *D-restriction-process$_{ptick}$*, *elim disjE conjE*)
(*solves simp*, *use front-tickFree-Nil* **in** *blast*,
metis (*no-types*) *T-imp-front-tickFree append-self-conv front-tickFree-nonempty-append-imp id-take-nth-drop is-processT3-TR-append leD length-take min.absorb4 take-all-iff*)

lemma *F-restriction-process$_{ptick}$I* :
$\langle(t,\ X) \in \mathcal{F}\ P \vee t \in \mathcal{T}\ P \wedge$ (*length* $t = n \wedge$ *tF* $t \vee n <$ *length* $t) \Longrightarrow (t,\ X) \in \mathcal{F}\ (P \downarrow n)\rangle$
by (*metis* (*mono-tags*, *lifting*) *D-restriction-process$_{ptick}$I F-restriction-process$_{ptick}$ Un-iff is-processT8*)

lemma *T-restriction-process$_{ptick}$I* :
$\langle t \in \mathcal{T}\ P \vee t \in \mathcal{T}\ P \wedge$ (*length* $t = n \wedge$ *tF* $t \vee n <$ *length* $t) \Longrightarrow t \in \mathcal{T}\ (P \downarrow n)\rangle$
using *F-restriction-process$_{ptick}$I T-F-spec* **by** *blast*

lemma *F-restriction-process$_{ptick}$-Suc-length-iff-F* :
‹$(t,\ X) \in \mathcal{F}\ (P \downarrow Suc\ (length\ t)) \longleftrightarrow (t,\ X) \in \mathcal{F}\ P$›
and *D-restriction-process$_{ptick}$-Suc-length-iff-D* :
‹$t \in \mathcal{D}\ (P \downarrow Suc\ (length\ t)) \longleftrightarrow t \in \mathcal{D}\ P$›
and *T-restriction-process$_{ptick}$-Suc-length-iff-T* :
‹$t \in \mathcal{T}\ (P \downarrow Suc\ (length\ t)) \longleftrightarrow t \in \mathcal{T}\ P$›
by (*auto simp add*: *restriction-process$_{ptick}$-projs*)

lemma *length-less-in-F-restriction-process$_{ptick}$* :
‹$length\ t < n \Longrightarrow (t,\ X) \in \mathcal{F}\ (P \downarrow n) \Longrightarrow (t,\ X) \in \mathcal{F}\ P$›
by (*auto elim*: *F-restriction-process$_{ptick}$E*)

lemma *length-le-in-T-restriction-process$_{ptick}$* :
‹$length\ t \leq n \Longrightarrow t \in \mathcal{T}\ (P \downarrow n) \Longrightarrow t \in \mathcal{T}\ P$›
by (*auto elim*: *T-restriction-process$_{ptick}$E*)

lemma *length-less-in-D-restriction-process$_{ptick}$* :
‹$length\ t < n \Longrightarrow t \in \mathcal{D}\ (P \downarrow n) \Longrightarrow t \in \mathcal{D}\ P$›
by (*auto elim*: *D-restriction-process$_{ptick}$E*)

lemma *not-tickFree-in-F-restriction-process$_{ptick}$-iff* :
‹$length\ t \leq n \Longrightarrow \neg\ tF\ t \Longrightarrow (t,\ X) \in \mathcal{F}\ (P \downarrow n) \longleftrightarrow (t,\ X) \in \mathcal{F}$
P›
by (*auto simp add*: *F-restriction-process$_{ptick}$*)

lemma *not-tickFree-in-D-restriction-process$_{ptick}$-iff* :
‹$length\ t \leq n \Longrightarrow \neg\ tF\ t \Longrightarrow t \in \mathcal{D}\ (P \downarrow n) \longleftrightarrow t \in \mathcal{D}\ P$›
by (*auto simp add*: *D-restriction-process$_{ptick}$*)

end

lemma *front-tickFreeE* :
‹$[\![ftF\ t;\ tF\ t \Longrightarrow thesis;\ \bigwedge t'\ r.\ t = t' \ @\ [\checkmark(r)] \Longrightarrow tF\ t' \Longrightarrow thesis]\!]$
$\Longrightarrow thesis$›
by (*metis front-tickFree-append-iff nonTickFree-n-frontTickFree not-Cons-self2*)

1.3 Proof obligation

instance
proof *intro-classes*
fix $P\ Q$ :: ‹$('a,\ 'r)\ process_{ptick}$›
have ‹$P \downarrow 0 = \bot$› **by** (*simp add*: *BOT-iff-Nil-D D-restriction-process$_{ptick}$*)
thus ‹$P \downarrow 0 \sqsubseteq_{FD}\ Q \downarrow 0$› **by** *simp*
next
show ‹$P \downarrow n \downarrow m = P \downarrow min\ n\ m$› **for** P :: ‹$('a,\ 'r)\ process_{ptick}$›
and $n\ m$

proof (*rule Process-eq-optimizedI*)
show ‹$t \in \mathcal{D}\ (P \downarrow n \downarrow m) \Longrightarrow t \in \mathcal{D}\ (P \downarrow min\ n\ m)$› **for** t
by (*elim restriction-process$_{ptick}$-elims*)
(*auto simp add*: *D-restriction-process$_{ptick}$ intro*: *front-tickFree-append*)
next
show ‹$t \in \mathcal{D}\ (P \downarrow min\ n\ m) \Longrightarrow t \in \mathcal{D}\ (P \downarrow n \downarrow m)$› **for** t
by (*elim restriction-process$_{ptick}$-elims*)
(*auto simp add*: *restriction-process$_{ptick}$-projs min-def split*: *if-split-asm*)
next
fix $t\ X$ **assume** ‹$(t, X) \in \mathcal{F}\ (P \downarrow n \downarrow m)$› ‹$t \notin \mathcal{D}\ (P \downarrow n \downarrow m)$›
hence ‹$(t, X) \in \mathcal{F}\ P \wedge length\ t \leq min\ n\ m$›
by (*elim F-restriction-process$_{ptick}$E*) (*auto simp add*: *restriction-process$_{ptick}$-projs*)
thus ‹$(t, X) \in \mathcal{F}\ (P \downarrow min\ n\ m)$› **unfolding** *F-restriction-process$_{ptick}$*
by *blast*
next
fix $t\ X$ **assume** ‹$(t, X) \in \mathcal{F}\ (P \downarrow min\ n\ m)$› ‹$t \notin \mathcal{D}\ (P \downarrow min\ n\ m)$›
hence ‹$(t, X) \in \mathcal{F}\ P \wedge length\ t \leq min\ n\ m$›
by (*elim F-restriction-process$_{ptick}$E*) (*auto simp add*: *restriction-process$_{ptick}$-projs*)
thus ‹$(t, X) \in \mathcal{F}\ (P \downarrow n \downarrow m)$› **unfolding** *F-restriction-process$_{ptick}$*
by *blast*
qed
next
show ‹$P \sqsubseteq_{FD}\ Q \Longrightarrow P \downarrow n \sqsubseteq_{FD}\ Q \downarrow n$› **for** $P\ Q :: ‹('a, 'r)\ process_{ptick}›$ **and** n
by (*simp add*: *refine-defs restriction-process$_{ptick}$-projs flip*: *T-F-spec*) *blast*
next
fix $P\ Q :: ‹('a, 'r)\ process_{ptick}›$ **assume** ‹$\neg\ P \sqsubseteq_{FD}\ Q$›
then consider t **where** ‹$t \in \mathcal{D}\ Q$› ‹$t \notin \mathcal{D}\ P$›
| $t\ X$ **where** ‹$(t, X) \in \mathcal{F}\ Q$› ‹$(t, X) \notin \mathcal{F}\ P$›
unfolding *refine-defs* **by** *auto*
thus ‹$\exists\, n.\ \neg\ P \downarrow n \sqsubseteq_{FD}\ Q \downarrow n$›
proof *cases*
fix t **assume** ‹$t \in \mathcal{D}\ Q$› ‹$t \notin \mathcal{D}\ P$›
with *D-restriction-process$_{ptick}$-Suc-length-iff-D*
have ‹$t \in \mathcal{D}\ (Q \downarrow Suc\ (length\ t)) \wedge t \notin \mathcal{D}\ (P \downarrow Suc\ (length\ t))$›
by *blast*
hence ‹$\neg\ P \downarrow Suc\ (length\ t) \sqsubseteq_{FD}\ Q \downarrow Suc\ (length\ t)$› **unfolding** *refine-defs* **by** *blast*
thus ‹$\exists\, n.\ \neg\ P \downarrow n \sqsubseteq_{FD}\ Q \downarrow n$› **..**
next
fix $t\ X$ **assume** ‹$(t, X) \in \mathcal{F}\ Q$› ‹$(t, X) \notin \mathcal{F}\ P$›
with *F-restriction-process$_{ptick}$-Suc-length-iff-F*
have ‹$(t, X) \in \mathcal{F}\ (Q \downarrow Suc\ (length\ t)) \wedge (t, X) \notin \mathcal{F}\ (P \downarrow Suc\ (length\ t))$› **by** *blast*

hence ‹¬ $P \downarrow Suc\ (length\ t) \sqsubseteq_{FD} Q \downarrow Suc\ (length\ t)$› **unfolding** *refine-defs* **by** *blast*
thus ‹$\exists n.\ \neg P \downarrow n \sqsubseteq_{FD} Q \downarrow n$› **..**
qed
qed

— Of course, we immediately recover the structure of *restriction-space*.

corollary ‹*OFCLASS*(('*a*, '*r*) $process_{ptick}$, *restriction-space-class*)›
by *intro-classes*

end

instance $process_{ptick}$ :: (*type*, *type*) *pcpo-restriction-space*
proof *intro-classes*
show ‹$P \downarrow 0 \sqsubseteq Q \downarrow 0$› **for** $P\ Q$ :: ‹('*a*, '*r*) $process_{ptick}$›
by (*metis below-refl restriction-0-related*)
next
show ‹$P \downarrow n \sqsubseteq Q \downarrow n$› **if** ‹$P \sqsubseteq Q$› **for** $P\ Q$ :: ‹('*a*, '*r*) $process_{ptick}$›
and *n*
proof (*unfold le-approx-def Refusals-after-def*, *safe*)
from ‹$P \sqsubseteq Q$›[*THEN le-approx1*] ‹$P \sqsubseteq Q$›[*THEN le-approx2T*]
show ‹$t \in \mathcal{D}\ (Q \downarrow n) \Longrightarrow t \in \mathcal{D}\ (P \downarrow n)$› **for** *t*
by (*simp add*: *D-restriction-process*$_{ptick}$ *subset-iff*) (*metis D-T*)
next
from ‹$P \sqsubseteq Q$›[*THEN le-approx2*] ‹$P \sqsubseteq Q$›[*THEN le-approx-lemma-T*]
show ‹$t \notin \mathcal{D}\ (P \downarrow n) \Longrightarrow (t, X) \in \mathcal{F}\ (P \downarrow n) \Longrightarrow (t, X) \in \mathcal{F}\ (Q \downarrow n)$›
and ‹$t \notin \mathcal{D}\ (P \downarrow n) \Longrightarrow (t, X) \in \mathcal{F}\ (Q \downarrow n) \Longrightarrow (t, X) \in \mathcal{F}\ (P \downarrow n)$› **for** *t X*
by (*auto simp add*: *restriction-process*$_{ptick}$*-projs*)
next
from ‹$P \sqsubseteq Q$›[*THEN le-approx3*] ‹$P \sqsubseteq Q$›[*THEN le-approx2T*]
show ‹$t \in$ *min-elems* $(\mathcal{D}\ (P \downarrow n)) \Longrightarrow t \in \mathcal{T}\ (Q \downarrow n)$› **for** *t*
by (*simp add*: *min-elems-def restriction-process*$_{ptick}$*-projs ball-Un subset-iff*)
(*metis is-processT7*)
qed
next
fix $P\ Q$ :: ‹('*a*, '*r*) $process_{ptick}$›
assume ‹$\neg P \sqsubseteq Q$›
then consider *t* **where** ‹$t \in \mathcal{D}\ Q$› ‹$t \notin \mathcal{D}\ P$›
| *t X* **where** ‹$t \notin \mathcal{D}\ P$› ‹$(t, X) \in \mathcal{F}\ P \longleftrightarrow (t, X) \notin \mathcal{F}\ Q$›
| *t* **where** ‹$t \in$ *min-elems* $(\mathcal{D}\ P)$› ‹$t \notin \mathcal{T}\ Q$›
unfolding *le-approx-def Refusals-after-def* **by** *blast*
thus ‹$\exists n.\ \neg P \downarrow n \sqsubseteq Q \downarrow n$›
proof *cases*
fix *t* **assume** ‹$t \in \mathcal{D}\ Q$› ‹$t \notin \mathcal{D}\ P$›

hence ‹$t \in \mathcal{D}\ (Q \downarrow Suc\ (length\ t))$› ‹$t \notin \mathcal{D}\ (P \downarrow Suc\ (length\ t))$›
by (*simp-all add*: *D-restriction-process*$_{ptick}$*-Suc-length-iff-D*)
hence ‹$\neg\ P \downarrow Suc\ (length\ t) \sqsubseteq Q \downarrow Suc\ (length\ t)$›
unfolding *le-approx-def* **by** *blast*
thus ‹$\exists n.\ \neg\ P \downarrow n \sqsubseteq Q \downarrow n$› **..**
next
fix $t\ X$ **assume** ‹$t \notin \mathcal{D}\ P$› ‹$(t, X) \in \mathcal{F}\ P \longleftrightarrow (t, X) \notin \mathcal{F}\ Q$›
hence ‹$t \notin \mathcal{D}\ (P \downarrow Suc\ (length\ t))$›
‹$(t, X) \in \mathcal{F}\ (P \downarrow Suc\ (length\ t)) \longleftrightarrow (t, X) \notin \mathcal{F}\ (Q \downarrow Suc\ (length\ t))$›
by (*simp-all add*: *D-restriction-process*$_{ptick}$*-Suc-length-iff-D*
F-restriction-process$_{ptick}$*-Suc-length-iff-F*)
hence ‹$\neg\ P \downarrow Suc\ (length\ t) \sqsubseteq Q \downarrow Suc\ (length\ t)$›
unfolding *le-approx-def Refusals-after-def* **by** *blast*
thus ‹$\exists n.\ \neg\ P \downarrow n \sqsubseteq Q \downarrow n$› **..**
next
fix t **assume** ‹$t \in min\text{-}elems\ (\mathcal{D}\ P)$› ‹$t \notin \mathcal{T}\ Q$›
hence ‹$t \in min\text{-}elems\ (\mathcal{D}\ (P \downarrow Suc\ (length\ t)))$› ‹$t \notin \mathcal{T}\ (Q \downarrow Suc\ (length\ t))$›
by (*simp-all add*: *min-elems-def D-restriction-process*$_{ptick}$*-Suc-length-iff-D*
T-restriction-process$_{ptick}$*-Suc-length-iff-T*)
(*meson length-less-in-D-restriction-process*$_{ptick}$ *less-SucI less-length-mono*)
hence ‹$\neg\ P \downarrow Suc\ (length\ t) \sqsubseteq Q \downarrow Suc\ (length\ t)$›
unfolding *le-approx-def* **by** *blast*
thus ‹$\exists n.\ \neg\ P \downarrow n \sqsubseteq Q \downarrow n$› **..**
qed
next
show ‹$chain\ S \Longrightarrow \exists P.\ range\ S <<|\ P$› **for** S :: ‹$nat \Rightarrow ('a, 'r)\ process_{ptick}$›
by (*simp add*: *cpo-class.cpo*)
next
show ‹$\exists P :: ('a, 'r)\ process_{ptick}.\ \forall Q.\ P \sqsubseteq Q$› **by** *blast*
qed

setup ‹*Sign.add-const-constraint* (***const-name***‹*restriction*›, *SOME* ***typ***‹$('a, 'r)\ process_{ptick} \Rightarrow nat \Rightarrow ('a, 'r)\ process_{ptick}$›)›
— Only allow (↓) for $('a, 'r)\ process_{ptick}$ (otherwise we would often have to specify).

1.4 Compatibility with Refinements

lemma *leF-restriction-process*$_{ptick}$*I*: ‹$P \downarrow n \sqsubseteq_F Q \downarrow n$›
if ‹$\bigwedge s\ X.\ (s, X) \in \mathcal{F}\ Q \Longrightarrow length\ s \leq n \Longrightarrow (s, X) \in \mathcal{F}\ (P \downarrow n)$›
proof (*unfold failure-refine-def*, *safe*)
show ‹$(s, X) \in \mathcal{F}\ (Q \downarrow n) \Longrightarrow (s, X) \in \mathcal{F}\ (P \downarrow n)$› **for** $s\ X$
proof (*elim F-restriction-process*$_{ptick}$*E exE conjE*)
show ‹$(s, X) \in \mathcal{F}\ Q \Longrightarrow length\ s \leq n \Longrightarrow (s, X) \in \mathcal{F}\ (P \downarrow n)$›
by (*simp add*: *F-restriction-process*$_{ptick}$) (*meson F-restriction-process*$_{ptick}$*E*

that)
next
fix s' t'
assume ‹$s = s' @ t'$› ‹$s' \in \mathcal{T}\ Q$› ‹*length* $s' = n$› ‹*tickFree* s'›
‹*front-tickFree* t'›
from ‹$s' \in \mathcal{T}\ Q$› ‹*length* $s' = n$› **have** ‹$s' \in \mathcal{T}\ P$›
by (*metis F-T T-F dual-order.refl length-le-in-T-restriction-process*$_{ptick}$
that)
with ‹$s = s' @ t'$› ‹*length* $s' = n$› ‹*tickFree* s'› ‹*front-tickFree* t'›
show ‹$(s, X) \in \mathcal{F}\ (P \downarrow n)$› **by** (*simp add*: *F-restriction-process*$_{ptick}$)
blast
qed
qed

lemma *leT-restriction-process*$_{ptick}I$: ‹$P \downarrow n \sqsubseteq_T\ Q \downarrow n$›
if ‹$\bigwedge s.\ s \in \mathcal{T}\ Q \Longrightarrow$ *length* $s \leq n \Longrightarrow s \in \mathcal{T}\ (P \downarrow n)$›
proof (*unfold trace-refine-def*, *safe*)
show ‹$s \in \mathcal{T}\ (Q \downarrow n) \Longrightarrow s \in \mathcal{T}\ (P \downarrow n)$› **for** s
proof (*elim T-restriction-process*$_{ptick}E$ *exE conjE*)
show ‹$s \in \mathcal{T}\ Q \Longrightarrow$ *length* $s \leq n \Longrightarrow s \in \mathcal{T}\ (P \downarrow n)$›
by (*simp add*: *T-restriction-process*$_{ptick}$) (*meson T-restriction-process*$_{ptick}E$
that)
next
fix s' t'
assume ‹$s = s' @ t'$› ‹$s' \in \mathcal{T}\ Q$› ‹*length* $s' = n$› ‹*tickFree* s'›
‹*front-tickFree* t'›
from ‹$s' \in \mathcal{T}\ Q$› ‹*length* $s' = n$› **have** ‹$s' \in \mathcal{T}\ P$›
using *length-le-in-T-restriction-process*$_{ptick}$ *that* **by** *blast*
with ‹$s = s' @ t'$› ‹*length* $s' = n$› ‹*tickFree* s'› ‹*front-tickFree* t'›
show ‹$s \in \mathcal{T}\ (P \downarrow n)$› **by** (*simp add*: *T-restriction-process*$_{ptick}$)
blast
qed
qed

lemma *leDT-restriction-process*$_{ptick}I$: ‹$P \downarrow n \sqsubseteq_{DT}\ Q \downarrow n$›
if ‹$\bigwedge s.\ s \in \mathcal{T}\ Q \Longrightarrow$ *length* $s \leq n \Longrightarrow s \in \mathcal{T}\ (P \downarrow n)$›
and ‹$\bigwedge s.$ *length* $s \leq n \Longrightarrow s \in \mathcal{D}\ Q \Longrightarrow s \in \mathcal{D}\ (P \downarrow n)$›
proof (*rule leD-leT-imp-leDT*)
show ‹$P \downarrow n \sqsubseteq_T\ Q \downarrow n$› **by** (*simp add*: *leT-restriction-process*$_{ptick}I$
that(*1*))
next
show ‹$P \downarrow n \sqsubseteq_D\ Q \downarrow n$›
proof (*unfold divergence-refine-def*, *rule subsetI*)
show ‹$s \in \mathcal{D}\ (Q \downarrow n) \Longrightarrow s \in \mathcal{D}\ (P \downarrow n)$› **for** s
proof (*elim D-restriction-process*$_{ptick}E$ *exE conjE*)
show ‹$s \in \mathcal{D}\ Q \Longrightarrow$ *length* $s \leq n \Longrightarrow s \in \mathcal{D}\ (P \downarrow n)$›
by (*simp add*: *D-restriction-process*$_{ptick}$) (*meson D-restriction-process*$_{ptick}E$

that(*2*))
next
fix s' t'
assume ‹$s = s' @ t'$› ‹$s' \in \mathcal{T}\ Q$› ‹*length* $s' = n$› ‹*tickFree* s'›
‹*front-tickFree* t'›
from ‹$s' \in \mathcal{T}\ Q$› ‹*length* $s' = n$› **have** ‹$s' \in \mathcal{T}\ P$›
using *length-le-in-T-restriction-process*$_{ptick}$ *that*(*1*) **by** *blast*
with ‹$s = s' @ t'$› ‹*length* $s' = n$› ‹*tickFree* s'› ‹*front-tickFree* t'›
show ‹$s \in \mathcal{D}\ (P \downarrow n)$› **by** (*simp add*: *D-restriction-process*$_{ptick}$)
blast
qed
qed
qed

lemma *leFD-restriction-process*$_{ptick}I$: ‹$P \downarrow n \sqsubseteq_{FD}\ Q \downarrow n$›
if ‹$\bigwedge s\ X.\ (s,\ X) \in \mathcal{F}\ Q \Longrightarrow$ *length* $s \leq n \Longrightarrow (s,\ X) \in \mathcal{F}\ (P \downarrow n)$›
and ‹$\bigwedge s.\ s \in \mathcal{D}\ Q \Longrightarrow$ *length* $s \leq n \Longrightarrow s \in \mathcal{D}\ (P \downarrow n)$›
proof (*rule leF-leD-imp-leFD*)
show ‹$P \downarrow n \sqsubseteq_F\ Q \downarrow n$› **by** (*simp add*: *leF-restriction-process*$_{ptick}I$
that(*1*))
next
show ‹$P \downarrow n \sqsubseteq_D\ Q \downarrow n$› **by** (*meson T-F-spec leDT-imp-leD leDT-restriction-process*$_{ptick}I$
that)
qed

1.5 First Laws

lemma *restriction-process*$_{ptick}$*-0* [*simp*] : ‹$P \downarrow 0 = \bot$›
by (*simp add*: *BOT-iff-Nil-D D-restriction-process*$_{ptick}$)

lemma *restriction-process*$_{ptick}$*-BOT* [*simp*] : ‹($\bot$ :: ('*a*, '*r*) *process*$_{ptick}$)
$\downarrow n = \bot$›
by (*simp add*: *BOT-iff-Nil-D D-restriction-process*$_{ptick}$ *D-BOT*)

lemma *restriction-process*$_{ptick}$*-is-BOT-iff* :
‹$P \downarrow n = \bot \longleftrightarrow n = 0 \vee P = \bot$›
by (*auto simp add*: *BOT-iff-Nil-D D-restriction-process*$_{ptick}$)

lemma *restriction-process*$_{ptick}$*-STOP* [*simp*] : ‹*STOP* $\downarrow n$ = (*if* $n =$
0 then $\bot$ *else STOP*)›
by (*simp add*: *STOP-iff-T T-restriction-process*$_{ptick}$ *T-STOP*)

lemma *restriction-process*$_{ptick}$*-is-STOP-iff* : ‹$P \downarrow n = STOP \longleftrightarrow n$
$\neq 0 \wedge P = STOP$›
by (*simp add*: *STOP-iff-T T-restriction-process*$_{ptick}$ *set-eq-iff*)
(*metis* (*no-types*, *lifting*) *append-self-conv2 front-tickFree-single gr0I*
less-numeral-extra(*3*) *list.discI list.size*(*3*) *tickFree-Nil*)

lemma *restriction-process$_{ptick}$-SKIP* [*simp*] : ‹*SKIP r* ↓ *n* = (*if n* = *0 then* ⊥ *else SKIP r*)›
by *simp* (*auto simp add*: *Process-eq-spec restriction-process$_{ptick}$-projs SKIP-projs*)

lemma *restriction-process$_{ptick}$-is-SKIP-iff* : ‹*P* ↓ *n* = *SKIP r* ⟷ *n* ≠ *0* ∧ *P* = *SKIP r*›
proof (*intro iffI conjI*)
show ‹*n* ≠ *0* ∧ *P* = *SKIP r* ⟹ *P* ↓ *n* = *SKIP r*› **by** *simp*
next
show ‹*P* ↓ *n* = *SKIP r* ⟹ *n* ≠ *0*› **by** (*metis restriction-process$_{ptick}$-0 SKIP-neq-BOT*)
next
show ‹*P* ↓ *n* = *SKIP r* ⟹ *P* = *SKIP r*›
by (*simp add*: *Process-eq-spec set-eq-iff SKIP-projs restriction-process$_{ptick}$-projs*, *safe*; *metis*)
qed

lemma *restriction-process$_{ptick}$-SKIPS* [*simp*] : ‹*SKIPS R* ↓ *n* = (*if n* = *0 then* ⊥ *else SKIPS R*)›
by *simp* (*auto simp add*: *Process-eq-spec restriction-process$_{ptick}$-projs SKIPS-projs*)

lemma *restriction-process$_{ptick}$-is-SKIPS-iff* : ‹*P* ↓ *n* = *SKIPS R* ⟷ *n* ≠ *0* ∧ *P* = *SKIPS R*›
proof (*cases* ‹*R* = {}›)
show ‹*R* = {} ⟹ *P* ↓ *n* = *SKIPS R* ⟷ *n* ≠ *0* ∧ *P* = *SKIPS R*›
by (*simp add*: *restriction-process$_{ptick}$-is-STOP-iff*)
next
show ‹*P* ↓ *n* = *SKIPS R* ⟷ *n* ≠ *0* ∧ *P* = *SKIPS R*› **if** ‹*R* ≠ {}›
proof (*intro iffI conjI*)
show ‹*n* ≠ *0* ∧ *P* = *SKIPS R* ⟹ *P* ↓ *n* = *SKIPS R*› **by** *simp*
next
show ‹*P* ↓ *n* = *SKIPS R* ⟹ *n* ≠ *0*›
by (*metis BOT-iff-Nil-D D-SKIPS empty-iff restriction-process$_{ptick}$-0*)
next
show ‹*P* ↓ *n* = *SKIPS R* ⟹ *P* = *SKIPS R*›
by (*simp add*: *Process-eq-spec* ‹*R* ≠ {}› *SKIPS-projs restriction-process$_{ptick}$-projs*, *safe*; *blast*)
qed
qed

1.6 Monotony

1.6.1 $P \downarrow n$ is an Approximation of the P

lemma *restriction-process$_{ptick}$-approx-self* : ‹*P* ↓ *n* ⊑ *P*›

proof (*unfold le-approx-def Refusals-after-def*, *safe*)
show ‹$t \in \mathcal{D}\ P \Longrightarrow t \in \mathcal{D}\ (P \downarrow n)$› **for** t **by** (*simp add*: *D-restriction-process*$_{ptick}$)
next
show ‹$t \notin \mathcal{D}\ (P \downarrow n) \Longrightarrow (t,\ X) \in \mathcal{F}\ (P \downarrow n) \Longrightarrow (t,\ X) \in \mathcal{F}\ P$›
for $t\ X$
by (*auto simp add*: *D-restriction-process*$_{ptick}$ *elim*: *F-restriction-process*$_{ptick}E$)
next
show ‹$t \notin \mathcal{D}\ (P \downarrow n) \Longrightarrow (t,\ X) \in \mathcal{F}\ P \Longrightarrow (t,\ X) \in \mathcal{F}\ (P \downarrow n)$›
for $t\ X$
by (*auto simp add*: *restriction-process*$_{ptick}$*-projs*)
next
show ‹$t \in$ *min-elems* $(\mathcal{D}\ (P \downarrow n)) \Longrightarrow t \in \mathcal{T}\ P$› **for** t
by (*auto simp add*: *min-elems-def D-restriction-process*$_{ptick}$ *ball-Un D-T*)
(*metis append.right-neutral front-tickFree-charn less-append nil-less2*)
qed

lemma *restriction-process*$_{ptick}$*-FD-self* : ‹$P \downarrow n \sqsubseteq_{FD} P$›
by (*simp add*: *le-approx-imp-le-ref restriction-process*$_{ptick}$*-approx-self*)

lemma *restriction-process*$_{ptick}$*-F-self* : ‹$P \downarrow n \sqsubseteq_{F} P$›
by (*simp add*: *restriction-process*$_{ptick}$*-FD-self leFD-imp-leF*)

lemma *restriction-process*$_{ptick}$*-D-self* : ‹$P \downarrow n \sqsubseteq_{D} P$›
by (*simp add*: *restriction-process*$_{ptick}$*-FD-self leFD-imp-leD*)

lemma *restriction-process*$_{ptick}$*-T-self* : ‹$P \downarrow n \sqsubseteq_{T} P$›
by (*simp add*: *restriction-process*$_{ptick}$*-F-self leF-imp-leT*)

lemma *restriction-process*$_{ptick}$*-DT-self* : ‹$P \downarrow n \sqsubseteq_{DT} P$›
by (*simp add*: *restriction-process*$_{ptick}$*-D-self restriction-process*$_{ptick}$*-T-self leD-leT-imp-leDT*)

1.6.2 Monotony of ($\downarrow$)

lemma *Suc-right-mono-restriction-process*$_{ptick}$: ‹$P \downarrow n \sqsubseteq P \downarrow$ *Suc* n›
by (*metis restriction-process*$_{ptick}$*-approx-self restriction-chainD restriction-chain-restrictions*)

lemma *Suc-right-mono-restriction-process*$_{ptick}$*-FD* : ‹$P \downarrow n \sqsubseteq_{FD} P \downarrow$ *Suc* n›
by (*simp add*: *Suc-right-mono-restriction-process*$_{ptick}$ *le-approx-imp-le-ref*)

lemma *Suc-right-mono-restriction-process*$_{ptick}$*-F* : ‹$P \downarrow n \sqsubseteq_{F} P \downarrow$ *Suc* n›
by (*simp add*: *Suc-right-mono-restriction-process*$_{ptick}$*-FD leFD-imp-leF*)

lemma *Suc-right-mono-restriction-process*$_{ptick}$*-D* : ‹$P \downarrow n \sqsubseteq_{D} P \downarrow$ *Suc* n›

by (*simp add*: *Suc-right-mono-restriction-process*$_{ptick}$*-FD leFD-imp-leD*)

lemma *Suc-right-mono-restriction-process*$_{ptick}$*-T* : ‹$P \downarrow n \sqsubseteq_T P \downarrow Suc\ n$›
by (*simp add*: *Suc-right-mono-restriction-process*$_{ptick}$*-FD leFD-imp-leF leF-imp-leT*)

lemma *Suc-right-mono-restriction-process*$_{ptick}$*-DT* : ‹$P \downarrow n \sqsubseteq_{DT} P \downarrow Suc\ n$›
by (*simp add*: *Suc-right-mono-restriction-process*$_{ptick}$*-D Suc-right-mono-restriction-process*$_{ptick}$*-T leD-leT-imp-leDT*)

lemma *le-right-mono-restriction-process*$_{ptick}$: ‹$n \leq m \Longrightarrow P \downarrow n \sqsubseteq P \downarrow m$›
by (*metis restriction-process*$_{ptick}$*-approx-self restriction-chain-def-ter restriction-chain-restrictions*)

lemma *le-right-mono-restriction-process*$_{ptick}$*-FD* : ‹$n \leq m \Longrightarrow P \downarrow n \sqsubseteq_{FD} P \downarrow m$›
by (*simp add*: *le-approx-imp-le-ref le-right-mono-restriction-process*$_{ptick}$)

lemma *le-right-mono-restriction-process*$_{ptick}$*-F* : ‹$n \leq m \Longrightarrow P \downarrow n \sqsubseteq_F P \downarrow m$›
by (*simp add*: *leFD-imp-leF le-right-mono-restriction-process*$_{ptick}$*-FD*)

lemma *restriction-process*$_{ptick}$*-le-right-mono-D* : ‹$n \leq m \Longrightarrow P \downarrow n \sqsubseteq_D P \downarrow m$›
by (*simp add*: *leFD-imp-leD le-right-mono-restriction-process*$_{ptick}$*-FD*)

lemma *restriction-process*$_{ptick}$*-le-right-mono-T* : ‹$n \leq m \Longrightarrow P \downarrow n \sqsubseteq_T P \downarrow m$›
by (*simp add*: *leF-imp-leT le-right-mono-restriction-process*$_{ptick}$*-F*)

lemma *restriction-process*$_{ptick}$*-le-right-mono-DT* : ‹$n \leq m \Longrightarrow P \downarrow n \sqsubseteq_{DT} P \downarrow m$›
by (*simp add*: *restriction-process*$_{ptick}$*-le-right-mono-D restriction-process*$_{ptick}$*-le-right-mono-T leD-leT-imp-leDT*)

1.6.3 Interpretations of Refinements

lemma *ex-not-restriction-leD* : ‹$\exists n.\ \neg\ P \downarrow n \sqsubseteq_D Q \downarrow n$› **if** ‹$\neg\ P \sqsubseteq_D Q$›
proof −
from ‹$\neg\ P \sqsubseteq_D Q$› **obtain** t **where** ‹$t \in \mathcal{D}\ Q$› ‹$t \notin \mathcal{D}\ P$›
unfolding *divergence-refine-def* **by** *blast*
hence ‹$t \in \mathcal{D}\ (Q \downarrow Suc\ (length\ t))$› ‹$t \notin \mathcal{D}\ (P \downarrow Suc\ (length\ t))$›
by (*simp-all add*: *D-restriction-process*$_{ptick}$*-Suc-length-iff-D*)

hence ‹¬ $P \downarrow Suc\ (length\ t) \sqsubseteq_D Q \downarrow Suc\ (length\ t)$›
unfolding *divergence-refine-def* **by** *blast*
thus ‹∃ n. ¬ $P \downarrow n \sqsubseteq_D Q \downarrow n$› **..**
qed

interpretation *PRS-leF* : *PreorderRestrictionSpace* ‹$(\downarrow)$› ‹$(\sqsubseteq_F)$›
proof *unfold-locales*
show ‹$P \downarrow 0 \sqsubseteq_F Q \downarrow 0$› **for** $P\ Q$ :: ‹$('a, 'r)\ process_{ptick}$› **by** *simp*
next
show ‹$P \sqsubseteq_F Q \Longrightarrow P \downarrow n \sqsubseteq_F Q \downarrow n$› **for** $P\ Q$ :: ‹$('a, 'r)\ process_{ptick}$›
and n
by (*simp add*: *failure-refine-def F-restriction-process*$_{ptick}$
flip: *T-F-spec*) *blast*
next
fix $P\ Q$ :: ‹$('a, 'r)\ process_{ptick}$› **assume** ‹¬ $P \sqsubseteq_F Q$›
then obtain $t\ X$ **where** ‹$(t, X) \in \mathcal{F}\ Q$› ‹$(t, X) \notin \mathcal{F}\ P$›
unfolding *failure-refine-def* **by** *auto*
with *F-restriction-process*$_{ptick}$*-Suc-length-iff-F*
have ‹$(t, X) \in \mathcal{F}\ (Q \downarrow Suc\ (length\ t)) \wedge (t, X) \notin \mathcal{F}\ (P \downarrow Suc\ (length\ t))$› **by** *blast*
hence ‹¬ $P \downarrow Suc\ (length\ t) \sqsubseteq_F Q \downarrow Suc\ (length\ t)$› **unfolding** *failure-refine-def* **by** *blast*
thus ‹∃ n. ¬ $P \downarrow n \sqsubseteq_F Q \downarrow n$› **..**
next
show ‹$P \sqsubseteq_F Q \Longrightarrow Q \sqsubseteq_F R \Longrightarrow P \sqsubseteq_F R$› **for** $P\ Q\ R$ :: ‹$('a, 'r)\ process_{ptick}$›
by (*fact trans-F*)
qed

interpretation *PRS-leT* : *PreorderRestrictionSpace* ‹$(\downarrow)$› ‹$(\sqsubseteq_T)$›
proof *unfold-locales*
show ‹$P \downarrow 0 \sqsubseteq_T Q \downarrow 0$› **for** $P\ Q$ :: ‹$('a, 'r)\ process_{ptick}$› **by** *simp*
next
show ‹$P \sqsubseteq_T Q \Longrightarrow P \downarrow n \sqsubseteq_T Q \downarrow n$› **for** $P\ Q$ :: ‹$('a, 'r)\ process_{ptick}$›
and n
by (*auto simp add*: *trace-refine-def T-restriction-process*$_{ptick}$)
next
fix $P\ Q$ :: ‹$('a, 'r)\ process_{ptick}$› **assume** ‹¬ $P \sqsubseteq_T Q$›
then obtain t **where** ‹$t \in \mathcal{T}\ Q$› ‹$t \notin \mathcal{T}\ P$›
unfolding *trace-refine-def* **by** *auto*
with *T-restriction-process*$_{ptick}$*-Suc-length-iff-T*
have ‹$t \in \mathcal{T}\ (Q \downarrow Suc\ (length\ t)) \wedge t \notin \mathcal{T}\ (P \downarrow Suc\ (length\ t))$› **by** *blast*
hence ‹¬ $P \downarrow Suc\ (length\ t) \sqsubseteq_T Q \downarrow Suc\ (length\ t)$› **unfolding** *trace-refine-def* **by** *blast*
thus ‹∃ n. ¬ $P \downarrow n \sqsubseteq_T Q \downarrow n$› **..**
next
show ‹$P \sqsubseteq_T Q \Longrightarrow Q \sqsubseteq_T R \Longrightarrow P \sqsubseteq_T R$› **for** $P\ Q\ R$ :: ‹$('a, 'r)$›

process$_{ptick}$›
by (*fact trans-T*)
qed

interpretation *PRS-leDT* : *PreorderRestrictionSpace* ‹(↓)› ‹($\sqsubseteq_{DT}$)›
proof *unfold-locales*
show ‹*P* ↓ *0* $\sqsubseteq_{DT}$ *Q* ↓ *0*› **for** *P Q* :: ‹('*a*, '*r*) *process*$_{ptick}$› **by** *simp*
next
show ‹*P* $\sqsubseteq_{DT}$ *Q* ⟹ *P* ↓ *n* $\sqsubseteq_{DT}$ *Q* ↓ *n*› **for** *P Q* :: ‹('*a*, '*r*) *process*$_{ptick}$› **and** *n*
by (*auto simp add*: *refine-defs restriction-process*$_{ptick}$*-projs*)
next
fix *P Q* :: ‹('*a*, '*r*) *process*$_{ptick}$› **assume** ‹¬ *P* $\sqsubseteq_{DT}$ *Q*›
hence ‹¬ *P* $\sqsubseteq_{D}$ *Q* ∨ ¬ *P* $\sqsubseteq_{T}$ *Q*› **unfolding** *trace-divergence-refine-def*
by *blast*
with *ex-not-restriction-leD PRS-leT.ex-not-restriction-related*
have ‹(∃ *n*. ¬ *P* ↓ *n* $\sqsubseteq_{D}$ *Q* ↓ *n*) ∨ (∃ *n*. ¬ *P* ↓ *n* $\sqsubseteq_{T}$ *Q* ↓ *n*)› **by**
blast
thus ‹∃ *n*. ¬ *P* ↓ *n* $\sqsubseteq_{DT}$ *Q* ↓ *n*›
unfolding *trace-divergence-refine-def* **by** *blast*
next
show ‹*P* $\sqsubseteq_{DT}$ *Q* ⟹ *Q* $\sqsubseteq_{DT}$ *R* ⟹ *P* $\sqsubseteq_{DT}$ *R*› **for** *P Q R* :: ‹('*a*,
'*r*) *process*$_{ptick}$›
by (*fact trans-DT*)
qed

1.7 Continuity

context begin

private lemma *chain-restriction-process*$_{ptick}$: ‹*chain Y* ⟹ *chain*
(λ*i*. *Y i* ↓ *n*)›
by (*simp add*: *mono-restriction-below po-class.chain-def*)

private lemma *cont-prem-restriction-process*$_{ptick}$:
‹(⨆ *i*. *Y i*) ↓ *n* = (⨆ *i*. *Y i* ↓ *n*)› (**is** ‹*?lhs* = *?rhs*›) **if** ‹*chain Y*›
proof (*rule Process-eq-optimizedI*)
show ‹*t* ∈ $\mathcal{D}$ *?lhs* ⟹ *t* ∈ $\mathcal{D}$ *?rhs*› **for** *t*
by (*auto simp add*: *limproc-is-thelub chain-restriction-process*$_{ptick}$
D-restriction-process$_{ptick}$ *LUB-projs* ‹*chain Y*›)
next
show ‹*t* ∈ $\mathcal{D}$ *?rhs* ⟹ *t* ∈ $\mathcal{D}$ *?lhs*› **for** *t*
by (*simp add*: *limproc-is-thelub chain-restriction-process*$_{ptick}$
D-restriction-process$_{ptick}$ *LUB-projs* ‹*chain Y*›)
(*metis D-T append-eq-append-conv is-processT3-TR-append*)
next
show ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs* ⟹ (*t*, *X*) ∈ $\mathcal{F}$ *?rhs*› **for** *t X*

by (*auto simp add*: *limproc-is-thelub chain-restriction-process*$_{ptick}$
F-restriction-process$_{ptick}$ *LUB-projs* ‹*chain Y*›)
next
show ‹(*s*, *X*) ∈ $\mathcal{F}$ *?rhs* ⟹ (*s*, *X*) ∈ $\mathcal{F}$ *?lhs*› **for** *s X*
by (*simp add*: *limproc-is-thelub chain-restriction-process*$_{ptick}$
F-restriction-process$_{ptick}$ *LUB-projs* ‹*chain Y*›)
(*metis F-T append-eq-append-conv is-processT3-TR-append*)
qed

lemma *restriction-process*$_{ptick}$*-cont* [*simp*] : ‹*cont* (λ*x*. *f x* ↓ *n*)› **if** ‹*cont f*›
proof (*rule contI2*)
show ‹*monofun* (λ*x*. *f x* ↓ *n*)›
by (*simp add*: *cont2monofunE mono-restriction-below monofunI* ‹*cont f*›)
next
show ‹*chain Y* ⟹ *f* (⨆ *i*. *Y i*) ↓ *n* ⊑ (⨆ *i*. *f* (*Y i*) ↓ *n)›* **for** *Y*
by (*simp add*: *ch2ch-cont cont2contlubE cont-prem-restriction-process*$_{ptick}$ ‹*cont f*›)
qed

end

1.8 Completeness

Processes are actually an instance of *complete-restriction-space*.

lemma *chain-restriction-chain* :
‹*restriction-chain σ* ⟹ *chain σ*› **for** *σ* :: ‹*nat* ⇒ ('*a*, '*r*) *process*$_{ptick}$›
by (*metis po-class.chainI restriction-process*$_{ptick}$*-approx-self restriction-chainD*)

lemma *restricted-LUB-restriction-chain-is* :
‹(λ*n*. (⨆ *n*. *σ n*) ↓ *n*) = *σ*› **if** ‹*restriction-chain σ*›
proof (*rule ext*)
have ‹*chain σ*› **by** (*simp add*: *chain-restriction-chain* ‹*restriction-chain σ*›)
moreover have ‹*σ* = (λ*n*. *σ n* ↓ *n*)›
by (*simp add*: *restricted-restriction-chain-is* ‹*restriction-chain σ*›)
ultimately have ‹*chain* (λ*n*. *σ n* ↓ *n*)› **by** *simp*

have ‹*length t* < *n* ⟹ *t* ∈ $\mathcal{D}$ (*σ n*) ⟷ (∀ *i*. *t* ∈ $\mathcal{D}$ (*σ i*))› **for** *t n*
proof *safe*
show ‹*t* ∈ $\mathcal{D}$ (*σ i*)› **if** ‹*length t* < *n*› ‹*t* ∈ $\mathcal{D}$ (*σ n*)› **for** *i*
proof (*cases* ‹*i* ≤ *n*›)
from ‹*t* ∈ $\mathcal{D}$ (*σ n*)› ‹*chain σ*› *le-approx-def po-class.chain-mono*
show ‹*i* ≤ *n* ⟹ *t* ∈ $\mathcal{D}$ (*σ i*)› **by** *blast*
next

```
    from ‹length t < n› ‹t ∈ 𝒟 (σ n)› show ‹¬ i ≤ n ⟹ t ∈ 𝒟 (σ
i)›
      by (induct n, simp-all)
      (metis ‹restriction-chain σ› length-less-in-D-restriction-process_ptick
          nat-le-linear restriction-chain-def-ter)
   qed
  next
   show ‹∀ i. t ∈ 𝒟 (σ i) ⟹ t ∈ 𝒟 (σ n)› by simp
  qed
  hence * : ‹length t < n ⟹ t ∈ 𝒟 (σ n) ⟷ t ∈ 𝒟 (⨆ i. σ i)› for
t n
   by (simp add: D-LUB ‹chain σ› limproc-is-thelub)

  show ‹(⨆ n. σ n) ↓ n = σ n› for n
  proof (subst (3) ‹σ = (λn. σ n ↓ n)›, rule Process-eq-optimizedI)
   show ‹t ∈ 𝒟 ((⨆ n. σ n) ↓ n) ⟹ t ∈ 𝒟 (σ n ↓ n)› for t
   proof (elim D-restriction-process_ptickE)
    show ‹t ∈ 𝒟 (⨆ n. σ n) ⟹ length t ≤ n ⟹ t ∈ 𝒟 (σ n ↓ n)›
    by (simp add: ‹chain σ› limproc-is-thelub D-LUB D-restriction-process_ptick)
   next
    show ‹⟦t = u @ v; u ∈ 𝒯 (⨆ n. σ n); length u = n; tF u; ftF v⟧
⟹ t ∈ 𝒟 (σ n ↓ n)› for u v
        by (auto simp add: ‹chain σ› limproc-is-thelub LUB-projs
restriction-process_ptick-projs)
   qed
  next
   fix t assume ‹t ∈ 𝒟 (σ n ↓ n)›
   hence ‹ftF t› by (simp add: D-imp-front-tickFree)
   with ‹t ∈ 𝒟 (σ n ↓ n)› consider ‹length t < n› ‹t ∈ 𝒟 (σ n)›
    | t' r where ‹t = t' @ [✓(r)]› ‹tF t'› ‹length t' < n› ‹t' ∈ 𝒟 (σ
n)›
    | u v  where ‹t = u @ v› ‹u ∈ 𝒯 (σ n)› ‹length u = n› ‹tF u›
‹ftF v›
    by (auto elim!: D-restriction-process_ptickE)
     (metis D-T Suc-le-lessD antisym-conv2 append.right-neutral
       front-tickFree-Nil front-tickFree-nonempty-append-imp
      is-processT9 length-append-singleton nonTickFree-n-frontTickFree)
   thus ‹t ∈ 𝒟 ((⨆ n. σ n) ↓ n)›
   proof cases
    show ‹length t < n ⟹ t ∈ 𝒟 (σ n) ⟹ t ∈ 𝒟 ((⨆ n. σ n) ↓ n)›
     by (simp add: D-restriction-process_ptick *)
   next
    fix t' r assume ‹t = t' @ [✓(r)]› ‹tF t'› ‹length t' < n› ‹t' ∈ 𝒟
(σ n)›
    from ‹length t' < n› ‹t' ∈ 𝒟 (σ n)› have ‹t' ∈ 𝒟 ((⨆ n. σ n) ↓
n)›
     by (simp add: D-restriction-process_ptick *)
    thus ‹t ∈ 𝒟 ((⨆ n. σ n) ↓ n)›
     by (simp add: ‹t = t' @ [✓(r)]› ‹tF t'› is-processT7)
```

```
    next
      fix u v assume ‹t = u @ v› ‹u ∈ 𝒯 (σ n)› ‹length u = n› ‹tF
u› ‹ftF v›
      from ‹length u = n› ‹u ∈ 𝒯 (σ n)› have ‹u ∈ 𝒯 (σ (Suc n))›
        by (metis length-le-in-T-restriction-process_ptick nat-le-linear
            restriction-chainD ‹restriction-chain σ›)
    from ‹chain σ› ‹u ∈ 𝒯 (σ (Suc n))› D-T le-approx2T po-class.chain-mono
      have ‹i ≤ Suc n ⟹ u ∈ 𝒯 (σ i)› for i by blast
      moreover have ‹Suc n < i ⟹ u ∈ 𝒯 (σ i)› for i
      by (subst T-restriction-process_ptick-Suc-length-iff-T[symmetric])
         (metis ‹length u = n› ‹u ∈ 𝒯 (σ (Suc n))›
           restriction-chain-def-bis ‹restriction-chain σ›)
      ultimately have ‹u ∈ 𝒯 (⨆ i. σ i)›
       by (metis T-LUB-2 ‹chain σ› limproc-is-thelub linorder-not-le)
      with ‹ftF v› ‹length u = n› ‹tF u› show ‹t ∈ 𝒟 ((⨆ n. σ n) ↓
n)›
        by (auto simp add: ‹t = u @ v› D-restriction-process_ptick)
    qed
  next
    show ‹(t, X) ∈ ℱ ((⨆ n. σ n) ↓ n) ⟹ t ∉ 𝒟 ((⨆ n. σ n) ↓ n)
⟹ (t, X) ∈ ℱ (σ n ↓ n)› for t X
     by (meson ‹chain σ› is-processT8 is-ub-thelub proc-ord2a restric-
tion-process_ptick-approx-self)
  next
    fix t X assume ‹(t, X) ∈ ℱ (σ n ↓ n)› ‹t ∉ 𝒟 (σ n ↓ n)›
    hence ‹length t ≤ n› ‹(t, X) ∈ ℱ (σ n)› ‹t ∉ 𝒟 (σ n)›
    by (auto elim!: F-restriction-process_ptickE simp add: D-restriction-process_ptick)
    thus ‹(t, X) ∈ ℱ ((⨆ i. σ i) ↓ n)›
      by (simp add: F-restriction-process_ptick)
       (meson ‹chain σ› is-ub-thelub le-approx2)
  qed
qed


instance process_ptick :: (type, type) complete-restriction-space
proof (intro-classes, rule restriction-convergentI)
  show ‹σ −↓→ (⨆ i. σ i)› if ‹restriction-chain σ› for σ :: ‹nat ⇒
('a, 'b) process_ptick›
  proof (subst restricted-LUB-restriction-chain-is[symmetric])
    from ‹restriction-chain σ› show ‹restriction-chain σ› .
  next
    from restriction-tendsto-restrictions
    show ‹(λn. (⨆ i. σ i) ↓ n) −↓→ (⨆ i. σ i)› .
  qed
qed
```

This is a very powerful result. Now we can write fixed-point equations for processes like $\upsilon\ X.\ f\ X$, providing the fact that f is *constructive*.

setup ‹*Sign.add-const-constraint* (***const-name*** ‹*restriction*›, *NONE*)›
— Back to normal.

2 Constructiveness of Prefixes

2.1 Equality

lemma *restriction-process*$_{ptick}$*-Mprefix* :
‹$\Box a \in A \to P\ a \downarrow n = (case\ n\ of\ 0 \Rightarrow \bot \mid Suc\ m \Rightarrow \Box a{\in}A \to (P\ a \downarrow m))$› (**is** ‹*?lhs* = *?rhs*›)
proof (*cases n*)
show ‹$n = 0 \Longrightarrow$ *?lhs* = *?rhs*› **by** *simp*
next
fix *m* **assume** ‹$n = Suc\ m$›
show ‹*?lhs* = *?rhs*›
proof (*rule Process-eq-optimizedI*)
show ‹$t \in \mathcal{D}$ *?lhs* $\Longrightarrow t \in \mathcal{D}$ *?rhs*› **for** *t*
by (*auto simp add*: ‹$n = Suc\ m$› *Mprefix-projs D-restriction-process*$_{ptick}$) *blast*
next
fix *t* **assume** ‹$t \in \mathcal{D}$ *?rhs*›
with *D-imp-front-tickFree* **obtain** *a t'*
where ‹$a \in A$› ‹$t = ev\ a\ \#\ t'$› ‹*ftF t'*› ‹$t' \in \mathcal{D}\ (P\ a \downarrow m)$›
by (*auto simp add*: ‹$n = Suc\ m$› *D-Mprefix*)
thus ‹$t \in \mathcal{D}$ *?lhs*›
by (*simp add*: ‹$n = Suc\ m$› *D-restriction-process*$_{ptick}$ *Mprefix-projs*)
(*metis append-Cons event*$_{ptick}$*.disc*(*1*) *length-Cons tickFree-Cons-iff*)
next
show ‹$(t, X) \in \mathcal{F}$ *?lhs* $\Longrightarrow (t, X) \in \mathcal{F}$ *?rhs*› **for** *t X*
by (*auto simp add*: ‹$n = Suc\ m$› *restriction-process*$_{ptick}$*-projs Mprefix-projs*)
next
show ‹$(t, X) \in \mathcal{F}$ *?rhs* $\Longrightarrow t \notin \mathcal{D}$ *?rhs* $\Longrightarrow (t, X) \in \mathcal{F}$ *?lhs*› **for** *t X*
by (*auto simp add*: ‹$n = Suc\ m$› *restriction-process*$_{ptick}$*-projs Mprefix-projs*)
qed
qed

lemma *restriction-process*$_{ptick}$*-Mndetprefix* :
‹$\sqcap a \in A \to P\ a \downarrow n = (case\ n\ of\ 0 \Rightarrow \bot \mid Suc\ m \Rightarrow \sqcap a{\in}A \to (P\ a \downarrow m))$› (**is** ‹*?lhs* = *?rhs*›)
proof (*cases n*)
show ‹$n = 0 \Longrightarrow$ *?lhs* = *?rhs*› **by** *simp*
next
fix *m* **assume** ‹$n = Suc\ m$›

show ‹*?lhs* = *?rhs*›
proof (*rule Process-eq-optimizedI*)
show ‹*t* ∈ $\mathcal{D}$ *?lhs* ⟹ *t* ∈ $\mathcal{D}$ *?rhs*› **for** *t*
by (*auto simp add*: ‹*n* = *Suc m*› *Mndetprefix-projs D-restriction-process*$_{ptick}$)
blast
next
fix *t* **assume** ‹*t* ∈ $\mathcal{D}$ *?rhs*›
with *D-imp-front-tickFree* **obtain** *a t*′
where ‹*a* ∈ *A*› ‹*t* = *ev a* # *t*′› ‹*ftF t*′› ‹*t*′ ∈ $\mathcal{D}$ (*P a* ↓ *m*)›
by (*auto simp add*: ‹*n* = *Suc m*› *D-Mndetprefix*′)
thus ‹*t* ∈ $\mathcal{D}$ *?lhs*›
by (*simp add*: ‹*n* = *Suc m*› *D-restriction-process*$_{ptick}$ *Mndetpre-*
fix-projs)
(*metis append-Cons event*$_{ptick}$*.disc*(*1*) *length-Cons tickFree-Cons-iff*)
next
show ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs* ⟹ (*t*, *X*) ∈ $\mathcal{F}$ *?rhs*› **for** *t X*
by (*auto simp add*: ‹*n* = *Suc m*› *restriction-process*$_{ptick}$*-projs*
Mndetprefix-projs split: *if-split-asm*)
next
show ‹(*t*, *X*) ∈ $\mathcal{F}$ *?rhs* ⟹ *t* ∉ $\mathcal{D}$ *?rhs* ⟹ (*t*, *X*) ∈ $\mathcal{F}$ *?lhs*› **for**
t X
by (*auto simp add*: ‹*n* = *Suc m*› *restriction-process*$_{ptick}$*-projs*
Mndetprefix-projs split: *if-split-asm*)
qed
qed

corollary *restriction-process*$_{ptick}$*-write0* :
‹*a* → *P* ↓ *n* = (*case n of* 0 ⇒ ⊥ | *Suc m* ⇒ *a* → (*P* ↓ *m*))›
unfolding *write0-def* **by** (*simp add*: *restriction-process*$_{ptick}$*-Mprefix*)

corollary *restriction-process*$_{ptick}$*-write* :
‹*c*!*a* → *P* ↓ *n* = (*case n of* 0 ⇒ ⊥ | *Suc m* ⇒ *c*!*a* → (*P* ↓ *m*))›
unfolding *write-def* **by** (*simp add*: *restriction-process*$_{ptick}$*-Mprefix*)

corollary *restriction-process*$_{ptick}$*-read* :
‹*c*?*a*∈*A* → *P a* ↓ *n* = (*case n of* 0 ⇒ ⊥ | *Suc m* ⇒ *c*?*a*∈*A* → (*P a*
↓ *m*))›
unfolding *read-def comp-def* **by** (*simp add*: *restriction-process*$_{ptick}$*-Mprefix*)

corollary *restriction-process*$_{ptick}$*-ndet-write* :
‹*c*!!*a*∈*A* → *P a* ↓ *n* = (*case n of* 0 ⇒ ⊥ | *Suc m* ⇒ *c*!!*a*∈*A* → (*P a*
↓ *m*))›
unfolding *ndet-write-def comp-def* **by** (*simp add*: *restriction-process*$_{ptick}$*-Mndetprefix*)

2.2 Constructiveness

lemma *Mprefix-constructive* : ‹*constructive* ($\lambda P.\ \Box a \in A \rightarrow P\ a$)›
by (*auto intro*: *constructiveI*
simp add: *restriction-process*$_{ptick}$*-Mprefix restriction-fun-def*)

lemma *Mndetprefix-constructive* : ‹*constructive* ($\lambda P.\ \sqcap a \in A \rightarrow P\ a$)›
by (*auto intro*: *constructiveI*
simp add: *restriction-process*$_{ptick}$*-Mndetprefix restriction-fun-def*)

lemma *write0-constructive* : ‹*constructive* ($\lambda P.\ a \rightarrow P$)›
by (*auto intro*: *constructiveI simp add*: *restriction-process*$_{ptick}$*-write0*)

lemma *write-constructive* : ‹*constructive* ($\lambda P.\ c!a \rightarrow P$)›
by (*auto intro*: *constructiveI simp add*: *restriction-process*$_{ptick}$*-write*)

lemma *read-constructive* : ‹*constructive* ($\lambda P.\ c?a \in A \rightarrow P\ a$)›
by (*auto intro*: *constructiveI*
simp add: *restriction-process*$_{ptick}$*-read restriction-fun-def*)

lemma *ndet-write-constructive* : ‹*constructive* ($\lambda P.\ c!!a \in A \rightarrow P\ a$)›
by (*auto intro*: *constructiveI*
simp add: *restriction-process*$_{ptick}$*-ndet-write restriction-fun-def*)

3 Non Destructiveness of Choices

3.1 Equality

lemma *restriction-process*$_{ptick}$*-Ndet* : ‹$P \sqcap Q \downarrow n = (P \downarrow n) \sqcap (Q \downarrow n)$›
by (*auto simp add*: *Process-eq-spec Ndet-projs restriction-process*$_{ptick}$*-projs*)

lemma *restriction-process*$_{ptick}$*-GlobalNdet* :
‹$(\sqcap a \in A.\ P\ a) \downarrow n = (\mathit{if}\ n = 0\ \mathit{then}\ \bot\ \mathit{else}\ \sqcap a \in A.\ (P\ a \downarrow n))$›
by *simp* (*auto simp add*: *Process-eq-spec GlobalNdet-projs restriction-process*$_{ptick}$*-projs*)

lemma *restriction-process*$_{ptick}$*-GlobalDet* :
‹$(\Box a \in A.\ P\ a) \downarrow n = (\mathit{if}\ n = 0\ \mathit{then}\ \bot\ \mathit{else}\ \Box a \in A.\ (P\ a \downarrow n))$›
(**is** ‹*?lhs* = (*if n = 0 then* ⊥ *else ?rhs*)›)
proof (*split if-split*, *intro conjI impI*)
show ‹$n = 0 \Longrightarrow$ *?lhs* = ⊥› **by** *simp*
next
show ‹*?lhs* = *?rhs*› **if** ‹$n \neq 0$›
proof (*rule Process-eq-optimized-bisI*)
show ‹$t \in \mathcal{D}$ *?lhs* $\Longrightarrow t \in \mathcal{D}$ *?rhs*› **for** *t*

by (*auto simp add*: *GlobalDet-projs D-restriction-process*$_{ptick}$ ‹$n \neq 0$› *split*: *if-split-asm*)
next
show ‹$t \in \mathcal{D}$ *?rhs* $\Longrightarrow$ $t \in \mathcal{D}$ *?lhs*› **for** t
by (*auto simp add*: *GlobalDet-projs D-restriction-process*$_{ptick}$)
next
show ‹$([], X) \in \mathcal{F}$ *?lhs* $\Longrightarrow$ $([], X) \in \mathcal{F}$ *?rhs*› **for** X
by (*auto simp add*: *restriction-process*$_{ptick}$*-projs F-GlobalDet*)
next
show ‹$([], X) \in \mathcal{F}$ *?rhs* $\Longrightarrow$ $([], X) \in \mathcal{F}$ *?lhs*› **for** X
by (*auto simp add*: *restriction-process*$_{ptick}$*-projs F-GlobalDet*)
(*metis append-eq-Cons-conv event*$_{ptick}$*.disc*(*2*) *tickFree-Cons-iff*)
next
show ‹$(e \# t, X) \in \mathcal{F}$ *?lhs* $\Longrightarrow$ $(e \# t, X) \in \mathcal{F}$ *?rhs*› **for** e t X
by (*auto simp add*: ‹$n \neq 0$› *F-restriction-process*$_{ptick}$ *GlobalDet-projs split*: *if-split-asm*)
next
show ‹$(e \# t, X) \in \mathcal{F}$ *?rhs* $\Longrightarrow$ $(e \# t, X) \in \mathcal{F}$ *?lhs*› **for** e t X
by (*auto simp add*: *F-restriction-process*$_{ptick}$ *GlobalDet-projs*)
qed
qed

lemma *restriction-process*$_{ptick}$*-Det*: ‹$P \Box Q \downarrow n = (P \downarrow n) \Box (Q \downarrow n)$› (**is** ‹*?lhs* = *?rhs*›)
proof −
have ‹$P \Box Q \downarrow n = \Box i \in \{0, 1 :: nat\}.\ (if\ i = 0\ then\ P\ else\ Q) \downarrow n$›
by (*simp add*: *GlobalDet-distrib-unit-bis*)
also have ‹$\ldots = (if\ n = 0\ then\ \bot\ else\ \Box i \in \{0, 1 :: nat\}.\ (if\ i = 0\ then\ P \downarrow n\ else\ Q \downarrow n))$›
by (*simp add*: *restriction-process*$_{ptick}$*-GlobalDet if-distrib if-distribR*)
also have ‹$\ldots = (P \downarrow n) \Box (Q \downarrow n)$› **by** (*simp add*: *GlobalDet-distrib-unit-bis*)
finally show ‹$P \Box Q \downarrow n = (P \downarrow n) \Box (Q \downarrow n)$› **.**
qed

corollary *restriction-process*$_{ptick}$*-Sliding*: ‹$P \rhd Q \downarrow n = (P \downarrow n) \rhd (Q \downarrow n)$›
by (*simp add*: *restriction-process*$_{ptick}$*-Det restriction-process*$_{ptick}$*-Ndet Sliding-def*)

3.2 Non Destructiveness

lemma *GlobalNdet-non-destructive* : ‹*non-destructive* ($\lambda P.\ \sqcap a \in A.\ P\ a$)›
by (*auto intro*: *non-destructiveI*
simp add: *restriction-process*$_{ptick}$*-GlobalNdet restriction-fun-def*)

lemma *Ndet-non-destructive* : ‹*non-destructive* (λ(*P*, *Q*). *P* ⊓ *Q*)›
by (*auto intro*: *non-destructiveI*
simp add: *restriction-process*$_{ptick}$*-Ndet restriction-prod-def*)

lemma *GlobalDet-non-destructive* : ‹*non-destructive* (λ*P*. □*a* ∈ *A*. *P* *a*)›
by (*auto intro*: *non-destructiveI*
simp add: *restriction-process*$_{ptick}$*-GlobalDet restriction-fun-def*)

lemma *Det-non-destructive* : ‹*non-destructive* (λ(*P*, *Q*). *P* □ *Q*)›
by (*auto intro*: *non-destructiveI*
simp add: *restriction-process*$_{ptick}$*-Det restriction-prod-def*)

corollary *Sliding-non-destructive* : ‹*non-destructive* (λ(*P*, *Q*). *P* ▷ *Q*)›
by (*auto intro*: *non-destructiveI*
simp add: *restriction-process*$_{ptick}$*-Sliding restriction-prod-def*)

4 Non Destructiveness of Renaming

4.1 Equality

lemma *restriction-process*$_{ptick}$*-Renaming*:
‹*Renaming P f g* ↓ *n* = *Renaming* (*P* ↓ *n*) *f g*› (**is** ‹*?lhs* = *?rhs*›)
proof (*rule Process-eq-optimizedI*)
show ‹*t* ∈ 𝒟 *?lhs* ⟹ *t* ∈ 𝒟 *?rhs*› **for** *t*
by (*auto simp add*: *Renaming-projs D-restriction-process*$_{ptick}$)
(*metis append.right-neutral front-tickFree-Nil map-event*$_{ptick}$*-tickFree*,
use front-tickFree-append **in** *blast*)
next
show ‹*t* ∈ 𝒟 *?rhs* ⟹ *t* ∈ 𝒟 *?lhs*› **for** *t*
by (*auto simp add*: *Renaming-projs D-restriction-process*$_{ptick}$
front-tickFree-append map-event$_{ptick}$*-tickFree*)
next
fix *t X* **assume** ‹(*t*, *X*) ∈ ℱ *?lhs*› ‹*t* ∉ 𝒟 *?lhs*›
then obtain *u* **where** ‹(*u*, *map-event*$_{ptick}$ *f g* −‘ *X*) ∈ ℱ *P*› ‹*t* = *map* (*map-event*$_{ptick}$ *f g*) *u*›
by (*simp add*: *Renaming-projs restriction-process*$_{ptick}$*-projs*) *blast*
thus ‹(*t*, *X*) ∈ ℱ *?rhs*› **by** (*auto simp add*: *F-Renaming F-restriction-process*$_{ptick}$)
next
fix *t X* **assume** ‹(*t*, *X*) ∈ ℱ *?rhs*› ‹*t* ∉ 𝒟 *?rhs*›
then obtain *u* **where** ‹(*u*, *map-event*$_{ptick}$ *f g* −‘ *X*) ∈ ℱ (*P* ↓ *n*)›
‹*t* = *map* (*map-event*$_{ptick}$ *f g*) *u*›
unfolding *Renaming-projs* **by** *blast*
from ‹(*u*, *map-event*$_{ptick}$ *f g* −‘ *X*) ∈ ℱ (*P* ↓ *n*)›
consider ‹*u* ∈ 𝒟 (*P* ↓ *n*)› | ‹(*u*, *map-event*$_{ptick}$ *f g* −‘ *X*) ∈ ℱ *P*›
unfolding *restriction-process*$_{ptick}$*-projs* **by** *blast*

thus ‹$(t,\ X) \in \mathcal{F}$ *?lhs*›
proof *cases*
assume ‹$u \in \mathcal{D}\ (P \downarrow n)$›
hence ‹$t \in \mathcal{D}$ *?rhs*›
proof (*elim D-restriction-process*$_{ptick}$*E*)
from ‹$t =$ *map* (*map-event*$_{ptick}$ *f g*) *u*› **show** ‹$u \in \mathcal{D}\ P \Longrightarrow t \in \mathcal{D}$ *?rhs*›
by (*cases* ‹*tF u*›, *simp-all add*: *D-Renaming D-restriction-process*$_{ptick}$)
(*use front-tickFree-Nil* **in** *blast*,
metis D-imp-front-tickFree butlast-snoc div-butlast-when-non-tickFree-iff front-tickFree-iff-tickFree-butlast front-tickFree-single map-append map-event$_{ptick}$*-front-tickFree nonTickFree-n-frontTickFree*)
next
show ‹$[\![u = v\ @\ w;\ v \in \mathcal{T}\ P;$ *length* $v = n;$ *tF* $v;$ *ftF* $w]\!] \Longrightarrow t \in \mathcal{D}$ *?rhs*› **for** *v w*
by (*simp add*: *D-Renaming D-restriction-process*$_{ptick}$ ‹$t =$ *map* (*map-event*$_{ptick}$ *f g*) *u*›)
(*use front-tickFree-Nil map-event*$_{ptick}$*-front-tickFree* **in** *blast*)
qed
with ‹$t \notin \mathcal{D}$ *?rhs*› **have** *False* **..**
thus ‹$(t,\ X) \in \mathcal{F}$ *?lhs*› **..**
next
show ‹$(u,$ *map-event*$_{ptick}$ *f g* $-`$ $X) \in \mathcal{F}\ P \Longrightarrow (t,\ X) \in \mathcal{F}$ (*Renaming P f g* $\downarrow$ *n*)›
by (*auto simp add*: *F-restriction-process*$_{ptick}$ *F-Renaming* ‹$t =$ *map* (*map-event*$_{ptick}$ *f g*) *u*›)
qed
qed

4.2 Non Destructiveness

lemma *Renaming-non-destructive* [*simp*] :
‹*non-destructive* ($\lambda P.$ *Renaming P f g*)›
by (*auto intro*: *non-destructiveI simp add*: *restriction-process*$_{ptick}$*-Renaming*)

5 Non Destructiveness of Sequential Composition

5.1 Refinement

lemma *restriction-process*$_{ptick}$*-Seq-FD* :
‹P **;** $Q \downarrow n \sqsubseteq_{FD} (P \downarrow n)$ **;** $(Q \downarrow n)$› (**is** ‹*?lhs* $\sqsubseteq_{FD}$ *?rhs*›)
proof −
have $*$: ‹$t \in \mathcal{D}\ (P \downarrow n) \Longrightarrow t \in \mathcal{D}$ *?lhs*› **for** *t*
by (*elim D-restriction-process*$_{ptick}$*E*)
(*auto simp add*: *Seq-projs D-restriction-process*$_{ptick}$)
{ fix *t u v r w x*

assume ‹u @ [✓(r)] ∈ $\mathcal{T}$ P› ‹*length* u < n› ‹v = w @ x› ‹w ∈ $\mathcal{T}$ Q›
‹*length* w = n› ‹*tF* w› ‹*ftF* x› ‹t = u @ v›
hence ‹t = (u @ *take* (n − *length* u) w) @ *drop* (n − *length* u) w @ x ∧
u @ *take* (n − *length* u) w ∈ $\mathcal{T}$ (P ; Q) ∧
length (u @ *take* (n − *length* u) w) = n ∧
tF (u @ *take* (n − *length* u) w) ∧ *ftF* (*drop* (n − *length* u) w @ x)›
by (*simp add*: ‹t = u @ v› *T-Seq*)
(*metis append-T-imp-tickFree append-take-drop-id front-tickFree-append is-processT3-TR-append list.distinct*(1) *tickFree-append-iff*)
with *D-restriction-process*$_{ptick}$ **have** ‹t ∈ $\mathcal{D}$ *?lhs*› **by** *blast*
} **note** ∗∗ = *this*

show ‹*?lhs* $\sqsubseteq_{FD}$ *?rhs*›
proof (*unfold refine-defs*, *safe*)
show *div* : ‹t ∈ $\mathcal{D}$ *?lhs*› **if** ‹t ∈ $\mathcal{D}$ *?rhs*› **for** t
proof −
from ‹t ∈ $\mathcal{D}$ *?rhs*› **consider** ‹t ∈ $\mathcal{D}$ (P ↓ n)›
| u v r **where** ‹t = u @ v› ‹u @ [✓(r)] ∈ $\mathcal{T}$ (P ↓ n)› ‹v ∈ $\mathcal{D}$ (Q ↓ n)›
unfolding *D-Seq* **by** *blast*
thus ‹t ∈ $\mathcal{D}$ *?lhs*›
proof *cases*
show ‹t ∈ $\mathcal{D}$ (P ↓ n) ⟹ t ∈ $\mathcal{D}$ *?lhs*› **by** (*fact* ∗)
next
fix u v r **assume** ‹t = u @ v› ‹u @ [✓(r)] ∈ $\mathcal{T}$ (P ↓ n)› ‹v ∈ $\mathcal{D}$ (Q ↓ n)›
from ‹u @ [✓(r)] ∈ $\mathcal{T}$ (P ↓ n)› **consider** ‹u @ [✓(r)] ∈ $\mathcal{D}$ (P ↓ n)› | ‹u @ [✓(r)] ∈ $\mathcal{T}$ P› ‹*length* u < n›
by (*elim T-restriction-process*$_{ptick}$*E*) (*auto simp add*: *D-restriction-process*$_{ptick}$)
thus ‹t ∈ $\mathcal{D}$ *?lhs*›
proof *cases*
show ‹u @ [✓(r)] ∈ $\mathcal{D}$ (P ↓ n) ⟹ t ∈ $\mathcal{D}$ *?lhs*›
by (*metis* ∗ *D-imp-front-tickFree* ‹t = u @ v› ‹v ∈ $\mathcal{D}$ (Q ↓ n)› *front-tickFree-append-iff is-processT7 is-processT9 not-Cons-self*)
next
from ‹v ∈ $\mathcal{D}$ (Q ↓ n)› **show** ‹u @ [✓(r)] ∈ $\mathcal{T}$ P ⟹ *length* u < n ⟹ t ∈ $\mathcal{D}$ *?lhs*›
proof (*elim D-restriction-process*$_{ptick}$*E exE conjE*)
show ‹u @ [✓(r)] ∈ $\mathcal{T}$ P ⟹ v ∈ $\mathcal{D}$ Q ⟹ t ∈ $\mathcal{D}$ *?lhs*›
by (*simp add*: ‹t = u @ v› *D-restriction-process*$_{ptick}$ *D-Seq*) *blast*
next
show ‹⟦u @ [✓(r)] ∈ $\mathcal{T}$ P; *length* u < n; v = w @ x; w ∈ $\mathcal{T}$ Q;

length $w = n$; *tF* w; *ftF* x⟧ ⟹ $t \in \mathcal{D}$ *?lhs*› **for** w x
using ** ‹$t = u$ @ v› **by** *blast*
qed
qed
qed
qed

have *mono* : ‹$(P \downarrow n)$; $(Q \downarrow n) \sqsubseteq P$; Q›
by (*simp add*: *fun-below-iff mono-Seq restriction-fun-def*
restriction-process$_{ptick}$*-approx-self*)

show ‹$(t, X) \in \mathcal{F}$ *?lhs*› **if** ‹$(t, X) \in \mathcal{F}$ *?rhs*› **for** t X
by (*meson F-restriction-process*$_{ptick}$*I div is-processT8 mono proc-ord2a that*)
qed
qed

corollary *restriction-process*$_{ptick}$*-MultiSeq-FD* :
‹$(SEQ\ l \in@\ L.\ P\ l) \downarrow n \sqsubseteq_{FD} SEQ\ l \in@\ L.\ (P\ l \downarrow n)$›
proof (*induct L rule*: *rev-induct*)
show ‹$(SEQ\ l \in@\ [].\ P\ l) \downarrow n \sqsubseteq_{FD} SEQ\ l \in@\ [].\ (P\ l \downarrow n)$› **by** *simp*
next
fix a L
assume *hyp*: ‹$(SEQ\ l \in@\ L.\ P\ l) \downarrow n \sqsubseteq_{FD} SEQ\ l \in@\ L.\ (P\ l \downarrow n)$›
have ‹$(SEQ\ l \in@\ (L\ @\ [a]).\ P\ l) \downarrow n = (SEQ\ l \in@\ L.\ P\ l$; $P\ a) \downarrow n$› **by** *simp*
also have ‹$\ldots \sqsubseteq_{FD} SEQ\ l \in@\ L.\ (P\ l \downarrow n)$; $(P\ a \downarrow n)$›
by (*fact trans-FD*[*OF restriction-process*$_{ptick}$*-Seq-FD mono-Seq-FD*[*OF hyp idem-FD*]])
also have ‹$\ldots = SEQ\ l \in@ (L\ @\ [a]).\ (P\ l \downarrow n)$› **by** *simp*
finally show ‹$(SEQ\ l \in@\ (L\ @\ [a]).\ P\ l) \downarrow n \sqsubseteq_{FD} \ldots$› .
qed

5.2 Non Destructiveness

lemma *Seq-non-destructive* :
‹*non-destructive* ($\lambda(P :: ('a, 'r)\ process_{ptick},\ Q).\ P$; Q)›
proof (*rule order-non-destructiveI*, *clarify*)
fix $P\ P'\ Q\ Q' ::$ ‹$('a, 'r)\ process_{ptick}$› **and** n
assume ‹$(P, Q) \downarrow n = (P', Q') \downarrow n$› ‹$0 < n$›
hence ‹$P \downarrow n = P' \downarrow n$› ‹$Q \downarrow n = Q' \downarrow n$›
by (*simp-all add*: *restriction-prod-def*)
show ‹P ; $Q \downarrow n \sqsubseteq_{FD} P'$; $Q' \downarrow n$›
proof (*rule leFD-restriction-process*$_{ptick}$*I*)
show *div* : ‹$t \in \mathcal{D}\ (P'$; $Q') \Longrightarrow t \in \mathcal{D}\ (P$; $Q \downarrow n)$› **if** ‹*length* $t \leq n$› **for** t
proof (*unfold D-Seq*, *safe*)
show ‹$t \in \mathcal{D}\ P' \Longrightarrow t \in \mathcal{D}\ (P$; $Q \downarrow n)$›

```
      by (simp add: D-restriction-process_ptick Seq-projs)
        (metis (no-types, opaque-lifting) D-restriction-process_ptickE
          D-restriction-process_ptickI ‹P ↓ n = P' ↓ n›)
    next
      fix u r v assume ‹t = u @ v› ‹u @ [✓(r)] ∈ 𝒯 P'› ‹v ∈ 𝒟 Q'›
      from ‹t = u @ v› ‹length t ≤ n› consider ‹v = []› ‹length u =
n›
        | ‹u = []› ‹length v = n› | ‹length u < n› ‹length v < n›
        using nless-le by (cases u; cases v, auto)
      thus ‹u @ v ∈ 𝒟 (P ; Q ↓ n)›
      proof cases
        assume ‹v = []› ‹length u = n›
      from ‹u @ [✓(r)] ∈ 𝒯 P'› append-T-imp-tickFree is-processT3-TR-append
        have ‹tF u› ‹u ∈ 𝒯 P'› by auto
        from ‹u ∈ 𝒯 P'› ‹length u = n› ‹P ↓ n = P' ↓ n› have ‹u ∈
𝒯 P›
          by (metis T-restriction-process_ptickI less-or-eq-imp-le
              length-le-in-T-restriction-process_ptick)
        with ‹tF u› have ‹u ∈ 𝒯 (P ; Q)› by (simp add: T-Seq)
        with ‹length u = n› show ‹u @ v ∈ 𝒟 (P ; Q ↓ n)›
          by (simp add: ‹v = []› ‹tF u› D-restriction-process_ptickI)
      next
        assume ‹u = []› ‹length v = n›
        from ‹0 < n› ‹u @ [✓(r)] ∈ 𝒯 P'› ‹u = []› ‹P ↓ n = P' ↓ n›
        have ‹[✓(r)] ∈ 𝒯 P›
          by (cases n, simp-all)
            (metis Suc-leI T-restriction-process_ptickI length-Cons
            length-le-in-T-restriction-process_ptick list.size(3) zero-less-Suc)
        show ‹u @ v ∈ 𝒟 (P ; Q ↓ n)›
        proof (cases ‹tF v›)
          assume ‹tF v›
          have ‹v ∈ 𝒯 Q'› by (simp add: D-T ‹v ∈ 𝒟 Q'›)
          with ‹length v = n› ‹Q ↓ n = Q' ↓ n› have ‹v ∈ 𝒯 Q›
            unfolding restriction-fun-def
            by (metis T-restriction-process_ptickI less-or-eq-imp-le
                length-le-in-T-restriction-process_ptick)
          with ‹[✓(r)] ∈ 𝒯 P› have ‹v ∈ 𝒯 (P ; Q)›
            by (simp add: T-Seq) (metis append-Nil)
          with ‹length v = n› show ‹u @ v ∈ 𝒟 (P ; Q ↓ n)›
            by (simp add: ‹u = []› ‹tF v› D-restriction-process_ptickI)
        next
          assume ‹¬ tF v›
          with ‹u = []› ‹Q ↓ n = Q' ↓ n› ‹¬ tF v› ‹length t ≤ n›
            ‹t = u @ v› ‹v ∈ 𝒟 Q'› have ‹v ∈ 𝒟 Q›
        by (metis append-self-conv2 not-tickFree-in-D-restriction-process_ptick-iff)
          with ‹[✓(r)] ∈ 𝒯 P› have ‹v ∈ 𝒟 (P ; Q)›
            by (simp add: D-Seq) (metis append-Nil)
          thus ‹u @ v ∈ 𝒟 (P ; Q ↓ n)›
            by (simp add: D-restriction-process_ptickI ‹u = []›)
```

```
      qed
    next
      assume ‹length u < n› ‹length v < n›
      from ‹u @ [✓(r)] ∈ 𝒯 P'› ‹length u < n› ‹P ↓ n = P' ↓ n›
      have ‹u @ [✓(r)] ∈ 𝒯 P›
        by (metis length-le-in-T-restriction-process_ptick Suc-le-eq
            length-append-singleton T-restriction-process_ptick I)
      moreover from ‹v ∈ 𝒟 Q'› ‹length v < n› ‹Q ↓ n = Q' ↓ n›
      have ‹v ∈ 𝒟 Q›
     by (metis D-restriction-process_ptick I length-less-in-D-restriction-process_ptick)
      ultimately show ‹u @ v ∈ 𝒟 (P ; Q ↓ n)›
        by (auto simp add: D-restriction-process_ptick D-Seq)
    qed
  qed

  fix t X assume ‹(t, X) ∈ ℱ (P' ; Q')› ‹length t ≤ n›
  consider ‹t ∈ 𝒟 (P' ; Q')› | ‹(t, X ∪ range tick) ∈ ℱ P'› ‹tF t›
    | u r v where ‹t = u @ v› ‹u @ [✓(r)] ∈ 𝒯 P'› ‹(v, X) ∈ ℱ Q'›
    using ‹(t, X) ∈ ℱ (P' ; Q')› by (auto simp add: Seq-projs)
  thus ‹(t, X) ∈ ℱ (P ; Q ↓ n)›
  proof cases
    from div ‹length t ≤ n› D-F
    show ‹t ∈ 𝒟 (P' ; Q') ⟹ (t, X) ∈ ℱ (P ; Q ↓ n)› by blast
  next
    show ‹(t, X) ∈ ℱ (P ; Q ↓ n)› if ‹(t, X ∪ range tick) ∈ ℱ P'›
‹tF t›
    proof (cases ‹length t = n›)
      assume ‹length t = n›
      from ‹(t, X ∪ range tick) ∈ ℱ P'› have ‹t ∈ 𝒯 P'› by (simp
add: F-T)
      with ‹P ↓ n = P' ↓ n› ‹length t ≤ n› have ‹t ∈ 𝒯 P›
     by (metis T-restriction-process_ptick I length-le-in-T-restriction-process_ptick)
      with ‹tF t› have ‹t ∈ 𝒯 (P ; Q)› by (simp add: T-Seq)
      with ‹length t = n› ‹tF t› show ‹(t, X) ∈ ℱ (P ; Q ↓ n)›
        by (simp add: F-restriction-process_ptick I)
    next
      assume ‹length t ≠ n›
      with ‹length t ≤ n› have ‹length t < n› by linarith
      with ‹P ↓ n = P' ↓ n› ‹(t, X ∪ range tick) ∈ ℱ P'›
      have ‹(t, X ∪ range tick) ∈ ℱ P›
     by (metis F-restriction-process_ptick I length-less-in-F-restriction-process_ptick)
      with ‹tF t› show ‹(t, X) ∈ ℱ (P ; Q ↓ n)›
        by (simp add: F-restriction-process_ptick I F-Seq)
    qed
  next
    fix u r v assume ‹t = u @ v› ‹u @ [✓(r)] ∈ 𝒯 P'› ‹(v, X) ∈ ℱ
Q'›
    from ‹t = u @ v› ‹length t ≤ n› consider ‹v = []› ‹length u =
n›
```

```
    | ‹u = []› ‹length v = n› | ‹length u < n› ‹length v < n›
    using nless-le by (cases u; cases v, auto)
  thus ‹(t, X) ∈ ℱ (P ; Q ↓ n)›
  proof cases
    assume ‹v = []› ‹length u = n›
  from ‹u @ [✓(r)] ∈ 𝒯 P'› append-T-imp-tickFree is-processT3-TR-append
    have ‹tF u› ‹u ∈ 𝒯 P'› by auto
    from ‹u ∈ 𝒯 P'› ‹length u = n› ‹P ↓ n = P' ↓ n› have ‹u ∈
𝒯 P›
      by (metis T-restriction-process_ptickI less-or-eq-imp-le
          length-le-in-T-restriction-process_ptick)
    with ‹tF u› have ‹u ∈ 𝒯 (P ; Q)› by (simp add: T-Seq)
    with ‹length u = n› show ‹(t, X) ∈ ℱ (P ; Q ↓ n)›
      by (simp add: ‹v = []› ‹tF u› F-restriction-process_ptick)
        (use ‹t = u @ v› ‹tF u› ‹v = []› front-tickFree-Nil in blast)
  next
    assume ‹u = []› ‹length v = n›
    from ‹0 < n› ‹u @ [✓(r)] ∈ 𝒯 P'› ‹u = []› ‹P ↓ n = P' ↓ n›
    have ‹[✓(r)] ∈ 𝒯 P›
      by (cases n, simp-all)
       (metis Suc-leI T-restriction-process_ptickI length-Cons
       length-le-in-T-restriction-process_ptick list.size(3) zero-less-Suc)
    show ‹(t, X) ∈ ℱ (P ; Q ↓ n)›
    proof (cases ‹tF v›)
      assume ‹tF v›
      from F-T ‹(v, X) ∈ ℱ Q'› have ‹v ∈ 𝒯 Q'› by blast
      with ‹length v = n› ‹Q ↓ n = Q' ↓ n› have ‹v ∈ 𝒯 Q›
        unfolding restriction-fun-def
        by (metis T-restriction-process_ptickI less-or-eq-imp-le
            length-le-in-T-restriction-process_ptick)
      with ‹[✓(r)] ∈ 𝒯 P› have ‹v ∈ 𝒯 (P ; Q)›
        by (simp add: T-Seq) (metis append-Nil)
      with ‹length v = n› have ‹t ∈ 𝒟 (P ; Q ↓ n)›
     by (simp add: ‹u = []› ‹tF v› ‹t = u @ v› D-restriction-process_ptickI)
      with D-F show ‹(t, X) ∈ ℱ (P ; Q ↓ n)› by blast
    next
      assume ‹¬ tF v›
      with ‹u = []› ‹Q ↓ n = Q' ↓ n› ‹¬ tF v› ‹length t ≤ n›
        ‹t = u @ v› ‹(v, X) ∈ ℱ Q'› have ‹(v, X) ∈ ℱ Q›
     by (metis append-self-conv2 not-tickFree-in-F-restriction-process_ptick-iff)
      with ‹[✓(r)] ∈ 𝒯 P› have ‹(t, X) ∈ ℱ (P ; Q)›
      by (simp add: ‹t = u @ v› ‹u = []› F-Seq) (metis append-Nil)
      thus ‹(t, X) ∈ ℱ (P ; Q ↓ n)›
        by (simp add: F-restriction-process_ptickI)
    qed
  next
    assume ‹length u < n› ‹length v < n›
    from ‹u @ [✓(r)] ∈ 𝒯 P'› ‹length u < n› ‹P ↓ n = P' ↓ n›
    have ‹u @ [✓(r)] ∈ 𝒯 P›
```

by (*metis length-le-in-T-restriction-process*$_{ptick}$ *Suc-le-eq*
length-append-singleton T-restriction-process$_{ptick}$*I*)
moreover from ‹$(v, X) \in \mathcal{F}\ Q'$› ‹*length* $v < n$› ‹$Q \downarrow n = Q' \downarrow n$›
have ‹$(v, X) \in \mathcal{F}\ Q$›
by (*metis F-restriction-process*$_{ptick}$*I length-less-in-F-restriction-process*$_{ptick}$)
ultimately show ‹$(t, X) \in \mathcal{F}\ (P ; Q \downarrow n)$›
by (*auto simp add*: ‹$t = u\ @\ v$› *F-restriction-process*$_{ptick}$ *F-Seq*)
qed
qed
qed
qed

6 Non Destructiveness of Synchronization Product

6.1 Preliminaries

lemma *D-Sync-optimized* :
‹$\mathcal{D}\ (P\ [\![A]\!]\ Q) =$
$\{v\ @\ w\ |t\ u\ v\ w.\ tF\ v \wedge ftF\ w \wedge$
v *setinterleaves* $((t, u),$ *range tick* $\cup$ *ev* ‘ $A) \wedge$
$(t \in \mathcal{D}\ P \wedge u \in \mathcal{T}\ Q \vee t \in \mathcal{D}\ Q \wedge u \in \mathcal{T}\ P)\}$›
(**is** ‹- = *?rhs*›)
proof (*intro subset-antisym subsetI*)
show ‹$d \in$ *?rhs* $\Longrightarrow d \in \mathcal{D}\ (P\ [\![A]\!]\ Q)$› **for** d
by (*auto simp add*: *D-Sync*)
next
fix d **assume** ‹$d \in \mathcal{D}\ (P\ [\![A]\!]\ Q)$›
then obtain $t\ u\ v\ w$
where $*$: ‹$d = v\ @\ w$› ‹*ftF* w› ‹$tF\ v \vee w = []$›
‹v *setinterleaves* $((t, u),$ *range tick* $\cup$ *ev* ‘ $A)$›
‹$t \in \mathcal{D}\ P \wedge u \in \mathcal{T}\ Q \vee t \in \mathcal{D}\ Q \wedge u \in \mathcal{T}\ P$›
unfolding *D-Sync* **by** *blast*
show ‹$d \in$ *?rhs*›
proof (*cases* ‹$tF\ v$›)
from $*$ **show** ‹$tF\ v \Longrightarrow d \in$ *?rhs*› **by** *blast*
next
assume ‹$\neg\ tF\ v$›
with $*(1, 3)$ **have** ‹$w = []$› ‹$d = v$› **by** *simp-all*
from *D-imp-front-tickFree* ‹$d = v$› ‹$d \in \mathcal{D}\ (P\ [\![A]\!]\ Q)$›
have ‹*ftF* v› **by** *blast*
with ‹$\neg\ tF\ v$› **obtain** $r\ v'$ **where** ‹$v = v'\ @\ [\checkmark(r)]$›
by (*meson nonTickFree-n-frontTickFree*)
with $*(4)$ **obtain** $t'\ u'$

where $**$: ‹$t = t'$ @ [✓(r)]› ‹$u = u'$ @ [✓(r)]›
‹v' *setinterleaves* ((t', u'), *range tick* ∪ *ev* ‘ A)›
by (*simp add*: ‹$v = v'$ @ [✓(r)]›)
(*meson* *(5) *D-imp-front-tickFree SyncWithTick-imp-NTF T-imp-front-tickFree*)
have ‹$t' \in \mathcal{D}\ P \wedge u' \in \mathcal{T}\ Q \vee t' \in \mathcal{D}\ Q \wedge u' \in \mathcal{T}\ P$›
by (*metis* *(5) **(1,2) *is-processT3-TR-append is-processT9*)
with **(3) ‹$d = v$› ‹*ftF* v› ‹$v = v'$ @ [✓(r)]›
front-tickFree-nonempty-append-imp **show** ‹$d \in$ *?rhs*› **by** *blast*
qed
qed

lemma *tickFree-interleave-iff* :
‹t *setinterleaves* ((u, v), S) ⟹ *tF* t ⟷ *tF* $u \wedge$ *tF* v›
by (*induct* ‹(u, S, v)› *arbitrary*: t u v *rule*: *setinterleaving.induct*)
(*auto split*: *if-split-asm option.split-asm*)

lemma *interleave-subsetL* :
‹*tF* t ⟹ {a. *ev* $a \in$ *set* u} ⊆ A ⟹
t *setinterleaves* ((u, v), *range tick* ∪ *ev* ‘ A) ⟹ $t = v$›
for t u v :: ‹('a, 'r) *trace*$_{ptick}$›
proof (*induct* ‹(u, *range tick* ∪ *ev* ‘ A :: ('a, 'r) *refusal*$_{ptick}$, v)›
arbitrary: t u v *rule*: *setinterleaving.induct*)
case *1* **thus** *?case* **by** *simp*
next
case (*2 y v*) **thus** *?case* **by** (*auto simp add*: *image-iff split*: *if-split-asm*)
next
case (*3 x u*) **thus** *?case*
by (*simp add*: *image-iff subset-iff split*: *if-split-asm*)
(*metis* (*mono-tags*, *lifting*) *event*$_{ptick}$.*exhaust*)
next
case (*4 x u y v*)
from *4.prems* **show** *?case*
apply (*simp add*: *subset-iff split*: *if-split-asm*)
apply (*metis* (*no-types*, *lifting*) *4.hyps*(*1*) *Un-iff mem-Collect-eq subsetI tickFree-Cons-iff*)
apply (*metis* (*no-types*, *lifting*) *4.hyps*(*2,4*) *4.prems*(*2,3*) *SyncHd-Tl SyncSameHdTl list.sel*(*1*) *setinterleaving-sym tickFree-Cons-iff*)
by (*metis event*$_{ptick}$.*exhaust imageI rangeI*)+
qed

lemma *interleave-subsetR* :
‹*tF* t ⟹ {a. *ev* $a \in$ *set* v} ⊆ A ⟹
t *setinterleaves* ((u, v), *range tick* ∪ *ev* ‘ A) ⟹ $t = u$›
by (*simp add*: *interleave-subsetL setinterleaving-sym*)

lemma *interleave-imp-lengthLR-le* :
‹t *setinterleaves* ((u, v), S) ⟹

```
   length u ≤ length t ∧ length v ≤ length t›
  by (induct ‹(u, S, v)› arbitrary: t u v rule: setinterleaving.induct;
     simp split: if-split-asm; use nat-le-linear not-less-eq-eq in fastforce)

lemma interleave-le-prefixLR :
  ‹t setinterleaves ((u, v), S) ⟹ u' ≤ u ⟹ v' ≤ v ⟹
   (∃ t' ≤ t. ∃ v'' ≤ v'. t' setinterleaves ((u', v''), S)) ∨
   (∃ t' ≤ t. ∃ u'' ≤ u'. t' setinterleaves ((u'', v'), S))›
proof (induct ‹(u, S, v)›
    arbitrary: t u u' v v' rule: setinterleaving.induct)
  case 1
  then show ?case by simp
next
  case (2 y v)
  thus ?case by (simp split: if-split-asm)
      (metis si-empty1 insert-iff nil-le)
next
  case (3 x u)
  thus ?case by (simp split: if-split-asm)
      (metis si-empty1 insert-iff nil-le)
next
  case (4 x u y v)
  show ?case
  proof (cases ‹u' = [] ∨ v' = []›)
    show ‹u' = [] ∨ v' = [] ⟹ ?case› by force
  next
    assume ‹¬ (u' = [] ∨ v' = [])›
    with 4.prems(2, 3)
    obtain u'' v'' where ‹u' = x # u''› ‹u'' ≤ u› ‹v' = y # v''› ‹v''
≤ v›
      by (meson Prefix-Order.prefix-Cons)
    with 4.prems(1) consider (both-in)    t' where ‹x ∈ S› ‹y ∈ S›
‹x = y› ‹t = x # t'›
      ‹t' setinterleaves ((u, v), S)›
    |      (inR-mvL)    t' where ‹x ∉ S› ‹y ∈ S› ‹t = x # t'›
      ‹t' setinterleaves ((u, y # v), S)›
    |      (inL-mvR)    t' where ‹x ∈ S› ‹y ∉ S› ‹t = y # t'›
      ‹t' setinterleaves ((x # u, v), S)›
    |      (notin-mvL) t' where ‹x ∉ S› ‹y ∉ S› ‹t = x # t'›
      ‹t' setinterleaves ((u, y # v), S)›
    |      (notin-mvR) t' where ‹x ∉ S› ‹y ∉ S› ‹t = y # t'›
      ‹t' setinterleaves ((x # u, v), S)›
      by (auto split: if-split-asm)
    thus ?case
    proof cases
      case both-in
      from 4.hyps(1)[OF both-in(1−3, 5) ‹u'' ≤ u› ‹v'' ≤ v›]
      show ?thesis
      proof (elim disjE exE conjE)
```

fix t'' v'''
assume ‹$t'' \leq t'$› ‹$v''' \leq v''$› ‹t'' *setinterleaves* $((u'', v'''), S)$›
hence ‹$y \# t'' \leq t \wedge y \# v''' \leq v' \wedge$
$(y \# t'')$ *setinterleaves* $((u', y \# v'''), S)$›
by (*simp add*: ‹$u' = x \# u''$› ‹$v' = y \# v''$› *both-in*(*2*−*4*))
thus *?thesis* **by** *blast*
next
fix t'' u'''
assume ‹$t'' \leq t'$› ‹$u''' \leq u''$› ‹t'' *setinterleaves* $((u''', v''), S)$›
hence ‹$x \# t'' \leq t \wedge x \# u''' \leq u' \wedge$
$(x \# t'')$ *setinterleaves* $((x \# u''', v'), S)$›
by (*simp add*: ‹$u' = x \# u''$› ‹$v' = y \# v''$› *both-in*(*2*−*4*))
thus *?thesis* **by** *blast*
qed
next
case *inR-mvL*
from *4.hyps*(*5*)[*simplified*, *OF inR-mvL*(*1*, *2* ,*4*) ‹$u'' \leq u$› ‹$v' \leq y \# v$›]
show *?thesis*
proof (*elim disjE exE conjE*)
fix t'' v'''
assume ‹$t'' \leq t'$› ‹$v''' \leq v'$› ‹t'' *setinterleaves* $((u'', v'''), S)$›
hence ‹$x \# t'' \leq t \wedge v''' \leq v' \wedge$
$(x \# t'')$ *setinterleaves* $((u', v'''), S)$›
by (*cases* v''') (*simp-all add*: ‹$u' = x \# u''$› ‹$v' = y \# v''$› *inR-mvL*(*1*, *3*))
thus *?thesis* **by** *blast*
next
fix t'' u'''
assume ‹$t'' \leq t'$› ‹$u''' \leq u''$› ‹t'' *setinterleaves* $((u''', v'), S)$›
hence ‹$x \# t'' \leq t \wedge x \# u''' \leq u' \wedge$
$(x \# t'')$ *setinterleaves* $((x \# u''', v'), S)$›
by (*simp add*: ‹$u' = x \# u''$› ‹$v' = y \# v''$› *inR-mvL*(*1*, *3*))
thus *?thesis* **by** *blast*
qed
next
case *inL-mvR*
from *4.hyps*(*2*)[*OF inL-mvR*(*1*, *2*, *4*) ‹$u' \leq x \# u$› ‹$v'' \leq v$›]
show *?thesis*
proof (*elim disjE exE conjE*)
fix t'' v'''
assume ‹$t'' \leq t'$› ‹$v''' \leq v''$› ‹t'' *setinterleaves* $((u', v'''), S)$›
hence ‹$y \# t'' \leq t \wedge y \# v''' \leq v' \wedge$
$(y \# t'')$ *setinterleaves* $((u', y \# v'''), S)$›
by (*simp add*: ‹$u' = x \# u''$› ‹$v' = y \# v''$› *inL-mvR*(*2*, *3*))
thus *?thesis* **by** *blast*
next
fix t'' u'''
assume ‹$t'' \leq t'$› ‹$u''' \leq u'$› ‹t'' *setinterleaves* $((u''', v''), S)$›

```
      hence ‹y # t'' ≤ t ∧ u''' ≤ u' ∧
            (y # t'') setinterleaves ((u''', v'), S)›
        by (cases u''') (simp-all add: ‹u' = x # u''› ‹v' = y # v''›
inL-mvR(2, 3))
      thus ?thesis by blast
    qed
  next
    case notin-mvL
    from 4.hyps(3)[OF notin-mvL(1, 2, 4) ‹u'' ≤ u› ‹v' ≤ y # v›]
    show ?thesis
    proof (elim disjE exE conjE)
      fix t'' v'''
      assume ‹t'' ≤ t'› ‹v''' ≤ v'› ‹t'' setinterleaves ((u'', v'''), S)›
      hence ‹x # t'' ≤ t ∧ v''' ≤ v' ∧
            (x # t'') setinterleaves ((u', v'''), S)›
        by (cases v''') (simp-all add: ‹u' = x # u''› ‹v' = y # v''›
notin-mvL(1, 3))
      thus ?thesis by blast
    next
      fix t'' u'''
      assume ‹t'' ≤ t'› ‹u''' ≤ u''› ‹t'' setinterleaves ((u''', v'), S)›
      hence ‹x # t'' ≤ t ∧ x # u''' ≤ u' ∧
            (x # t'') setinterleaves ((x # u''', v'), S)›
       by (simp add: ‹u' = x # u''› ‹v' = y # v''› notin-mvL(1, 3))
      thus ?thesis by blast
    qed
  next
    case notin-mvR
    from 4.hyps(4)[OF notin-mvR(1, 2, 4) ‹u' ≤ x # u› ‹v'' ≤ v›]
    show ?thesis
    proof (elim disjE exE conjE)
      fix t'' v'''
      assume ‹t'' ≤ t'› ‹v''' ≤ v''› ‹t'' setinterleaves ((u', v'''), S)›
      hence ‹y # t'' ≤ t ∧ y # v''' ≤ v' ∧
            (y # t'') setinterleaves ((u', y # v'''), S)›
       by (simp add: ‹u' = x # u''› ‹v' = y # v''› notin-mvR(2, 3))
      thus ?thesis by blast
    next
      fix t'' u'''
      assume ‹t'' ≤ t'› ‹u''' ≤ u'› ‹t'' setinterleaves ((u''', v''), S)›
      hence ‹y # t'' ≤ t ∧ u''' ≤ u' ∧
            (y # t'') setinterleaves ((u''', v'), S)›
        by (cases u''') (simp-all add: ‹u' = x # u''› ‹v' = y # v''›
notin-mvR(2, 3))
      thus ?thesis by blast
    qed
  qed
 qed
qed
```

lemma *restriction-process$_{ptick}$-Sync-FD-div-oneside* :
assumes ‹*tF u*› ‹*ftF v*› ‹*t-P* ∈ $\mathcal{D}$ (*P* ↓ *n*)› ‹*t-Q* ∈ $\mathcal{T}$ (*Q* ↓ *n*)›
‹*u setinterleaves* ((*t-P*, *t-Q*), *range tick* ∪ *ev* ‘ *A*)›
shows ‹*u* @ *v* ∈ $\mathcal{D}$ (*P* ⟦*A*⟧ *Q* ↓ *n*)›
proof (*insert assms*(*3*, *4*), *elim D-restriction-process$_{ptick}$E T-restriction-process$_{ptick}$E*)
from *assms*(*1*, *2*, *5*) **show** ‹*t-P* ∈ $\mathcal{D}$ *P* ⟹ *t-Q* ∈ $\mathcal{T}$ *Q* ⟹ *u* @ *v*
∈ $\mathcal{D}$ (*P* ⟦*A*⟧ *Q* ↓ *n*)›
by (*auto simp add*: *D-restriction-process$_{ptick}$ D-Sync*)
next
fix *t-Q′ t-Q″*
assume ∗ : ‹*t-P* ∈ $\mathcal{D}$ *P*› ‹*length t-P* ≤ *n*› ‹*t-Q* = *t-Q′* @ *t-Q″*›
‹*t-Q′* ∈ $\mathcal{T}$ *Q*› ‹*length t-Q′* = *n*› ‹*tF t-Q′*› ‹*ftF t-Q″*›
from ‹*t-Q* = *t-Q′* @ *t-Q″*› **have** ‹*t-Q′* ≤ *t-Q*› **by** *simp*
from *interleave-le-right*[*OF assms*(*5*) *this*]
obtain *t-P′ t-P″ u′ u″*
where ∗∗ : ‹*u* = *u′* @ *u″*› ‹*t-P* = *t-P′* @ *t-P″*›
‹*u′ setinterleaves* ((*t-P′*, *t-Q′*), *range tick* ∪ *ev* ‘ *A*)›
by (*meson Prefix-Order.prefixE*)
from *assms*(*1*) ‹*u* = *u′* @ *u″*› **have** ‹*tF u′*› **by** *auto*
moreover from ∗(*1*,*4*) ∗∗(*2*,*3*) **have** ‹*u′* ∈ $\mathcal{T}$ (*P* ⟦*A*⟧ *Q*)›
by (*simp add*: *T-Sync*) (*metis D-T is-processT3-TR-append*)
moreover have ‹*length t-Q′* ≤ *length u′*›
using ∗∗(*3*) *interleave-imp-lengthLR-le* **by** *blast*
ultimately have ‹*u′* ∈ $\mathcal{D}$ (*P* ⟦*A*⟧ *Q* ↓ *n*)›
by (*metis* ∗(*5*) *D-restriction-process$_{ptick}$I nless-le*)
with ∗∗(*1*) *assms*(*1*, *2*) **show** ‹*u* @ *v* ∈ $\mathcal{D}$ (*P* ⟦*A*⟧ *Q* ↓ *n*)›
by (*metis is-processT7 tickFree-append-iff tickFree-imp-front-tickFree*)
next
fix *t-P′ t-P″*
assume ∗ : ‹*t-P* = *t-P′* @ *t-P″*› ‹*t-P′* ∈ $\mathcal{T}$ *P*› ‹*length t-P′* = *n*›
‹*tF t-P′*› ‹*ftF t-P″*› ‹*t-Q* ∈ $\mathcal{T}$ *Q*› ‹*length t-Q* ≤ *n*›
from ‹*t-P* = *t-P′* @ *t-P″*› **have** ‹*t-P′* ≤ *t-P*› **by** *simp*
from *interleave-le-left*[*OF assms*(*5*) *this*]
obtain *t-Q′ t-Q″ u′ u″*
where ∗∗ : ‹*u* = *u′* @ *u″*› ‹*t-Q* = *t-Q′* @ *t-Q″*›
‹*u′ setinterleaves* ((*t-P′*, *t-Q′*), *range tick* ∪ *ev* ‘ *A*)›
by (*meson Prefix-Order.prefixE*)
from *assms*(*1*) ‹*u* = *u′* @ *u″*› **have** ‹*tF u′*› **by** *auto*
moreover from ∗(*2*,*6*) ∗∗(*2*,*3*) **have** ‹*u′* ∈ $\mathcal{T}$ (*P* ⟦*A*⟧ *Q*)›
by (*simp add*: *T-Sync*) (*metis is-processT3-TR-append*)
moreover have ‹*length t-P′* ≤ *length u′*›
using ∗∗(*3*) *interleave-imp-lengthLR-le* **by** *blast*
ultimately have ‹*u′* ∈ $\mathcal{D}$ (*P* ⟦*A*⟧ *Q* ↓ *n*)›
by (*metis* ∗(*3*) *D-restriction-process$_{ptick}$I nless-le*)
with ∗∗(*1*) *assms*(*1*, *2*) **show** ‹*u* @ *v* ∈ $\mathcal{D}$ (*P* ⟦*A*⟧ *Q* ↓ *n*)›
by (*metis is-processT7 tickFree-append-iff tickFree-imp-front-tickFree*)
next

fix *t-P′ t-P″ t-Q′ t-Q″*
assume $: ‹*t-P* = *t-P′* @ *t-P″*› ‹*t-P′* ∈ $\mathcal{T}$ *P*› ‹*length t-P′* = *n*›
‹*tF t-P′*› ‹*ftF t-P″*› ‹*t-Q* = *t-Q′* @ *t-Q″*› ‹*t-Q′* ∈ $\mathcal{T}$ *Q*›
‹*length t-Q′* = *n*› ‹*tF t-Q′*› ‹*ftF t-Q″*›
from $(*1*, *6*) **have** ‹*t-P′* ≤ *t-P*› ‹*t-Q′* ≤ *t-Q*› **by** *simp-all*
from *interleave-le-prefixLR*[*OF assms*(*5*) *this*]
show ‹*u* @ *v* ∈ $\mathcal{D}$ (*P* ⟦*A*⟧ *Q* ↓ *n*)›
proof (*elim disjE conjE exE*)
fix *u′ t-Q‴* **assume** $$: ‹*u′* ≤ *u*› ‹*t-Q‴* ≤ *t-Q′*›
‹*u′ setinterleaves* ((*t-P′*, *t-Q‴*), *range tick* ∪ *ev* ‘ *A*)›
from $(*7*) $$(*2*) *is-processT3-TR* **have** ‹*t-Q‴* ∈ $\mathcal{T}$ *Q*› **by** *blast*
with $$(*3*) ‹*t-P′* ∈ $\mathcal{T}$ *P*› **have** ‹*u′* ∈ $\mathcal{T}$ (*P* ⟦*A*⟧ *Q*)›
by (*auto simp add*: *T-Sync*)
moreover have ‹*n* ≤ *length u′*›
using $(*3*) $$(*3*) *interleave-imp-lengthLR-le* **by** *blast*
ultimately have ‹*u′* ∈ $\mathcal{D}$ (*P* ⟦*A*⟧ *Q* ↓ *n*)›
by (*metis* $$(*1*) *D-restriction-process*$_{ptick}$*I Prefix-Order.prefixE*
assms(*1*) *nless-le tickFree-append-iff*)
thus ‹*u* @ *v* ∈ $\mathcal{D}$ (*P* ⟦*A*⟧ *Q* ↓ *n*)›
by (*metis* $$(*1*) *Prefix-Order.prefixE assms*(*1*,*2*) *is-processT7*
tickFree-append-iff tickFree-imp-front-tickFree)
next
fix *u′ t-P‴* **assume** $$: ‹*u′* ≤ *u*› ‹*t-P‴* ≤ *t-P′*›
‹*u′ setinterleaves* ((*t-P‴*, *t-Q′*), *range tick* ∪ *ev* ‘ *A*)›
from $(*2*) $$(*2*) *is-processT3-TR* **have** ‹*t-P‴* ∈ $\mathcal{T}$ *P*› **by** *blast*
with $$(*3*) ‹*t-Q′* ∈ $\mathcal{T}$ *Q*› **have** ‹*u′* ∈ $\mathcal{T}$ (*P* ⟦*A*⟧ *Q*)›
by (*auto simp add*: *T-Sync*)
moreover have ‹*n* ≤ *length u′*›
using $(*8*) $$(*3*) *interleave-imp-lengthLR-le* **by** *blast*
ultimately have ‹*u′* ∈ $\mathcal{D}$ (*P* ⟦*A*⟧ *Q* ↓ *n*)›
by (*metis* $$(*1*) *D-restriction-process*$_{ptick}$*I Prefix-Order.prefixE*
assms(*1*) *nless-le tickFree-append-iff*)
thus ‹*u* @ *v* ∈ $\mathcal{D}$ (*P* ⟦*A*⟧ *Q* ↓ *n*)›
by (*metis* $$(*1*) *Prefix-Order.prefixE assms*(*1*,*2*) *is-processT7*
tickFree-append-iff tickFree-imp-front-tickFree)
qed
qed

6.2 Refinement

lemma *restriction-process*$_{ptick}$*-Sync-FD* :
‹*P* ⟦*A*⟧ *Q* ↓ *n* ⊑$_{FD}$ (*P* ↓ *n*) ⟦*A*⟧ (*Q* ↓ *n*)› (**is** ‹*?lhs* ⊑$_{FD}$ *?rhs*›)
proof (*unfold refine-defs*, *safe*)
show ‹*t* ∈ $\mathcal{D}$ *?rhs* ⟹ *t* ∈ $\mathcal{D}$ *?lhs*› **for** *t*
by (*unfold D-Sync-optimized*, *safe*)
(*solves* ‹*simp add*: *restriction-process*$_{ptick}$*-Sync-FD-div-oneside*›,
metis Sync-commute restriction-process$_{ptick}$*-Sync-FD-div-oneside*)
thus ‹(*t*, *X*) ∈ $\mathcal{F}$ ((*P* ↓ *n*) ⟦*A*⟧ (*Q* ↓ *n*)) ⟹ (*t*, *X*) ∈ $\mathcal{F}$ (*P* ⟦*A*⟧ *Q*
↓ *n*)› **for** *t X*

by (*meson is-processT8 le-approx2 mono-Sync restriction-process$_{ptick}$-approx-self*)
qed

The equality does not hold in general, but we can establish it by adding an assumption over the strict alphabets of the processes.

lemma *strict-events-of-subset-restriction-process$_{ptick}$-Sync* :
‹$P \llbracket A \rrbracket Q \downarrow n = (P \downarrow n) \llbracket A \rrbracket (Q \downarrow n)$› (**is** ‹*?lhs* = *?rhs*›)
if ‹$\boldsymbol{\alpha}(P) \subseteq A \vee \boldsymbol{\alpha}(Q) \subseteq A$›
proof (*rule FD-antisym*)
show ‹*?lhs* $\sqsubseteq_{FD}$ *?rhs*› **by** (*fact restriction-process$_{ptick}$-Sync-FD*)
next
have *div* : ‹$t \in \mathcal{D}\ (P \llbracket A \rrbracket Q) \Longrightarrow t \in \mathcal{D}$ *?rhs*› **for** *t*
by (*auto simp add*: *D-Sync restriction-process$_{ptick}$-projs*)

{ fix *t u v* **assume** ‹$t = u$ @ v› ‹$u \in \mathcal{T}\ (P \llbracket A \rrbracket Q)$› ‹*length u* = *n*› ‹*tF u*› ‹*ftF v*›
from *this*(*2*) **consider** ‹$u \in \mathcal{D}\ (P \llbracket A \rrbracket Q)$›
| *t-P t-Q* **where** ‹*t-P* $\in \mathcal{T}$ *P*› ‹*t-Q* $\in \mathcal{T}$ *Q*›
‹*u setinterleaves* ((*t-P*, *t-Q*), *range tick* ∪ *ev* ‘ *A*)›
unfolding *Sync-projs* **by** *blast*
hence ‹*t* $\in \mathcal{D}$ *?rhs*›
proof *cases*
show ‹$u \in \mathcal{D}\ (P \llbracket A \rrbracket Q) \Longrightarrow t \in \mathcal{D}$ *?rhs*›
by (*simp add*: ‹*ftF v*› ‹$t = u$ @ v› ‹*tF u*› *div is-processT7*)
next
fix *t-P t-Q* **assume** ‹*t-P* $\in \mathcal{T}$ *P*› ‹*t-Q* $\in \mathcal{T}$ *Q*›
and *setinter* : ‹*u setinterleaves* ((*t-P*, *t-Q*), *range tick* ∪ *ev* ‘ *A*)›
consider ‹*t-P* $\in \mathcal{D}$ *P* ∨ *t-Q* $\in \mathcal{D}$ *Q*› | ‹*t-P* $\notin \mathcal{D}$ *P*› ‹*t-Q* $\notin \mathcal{D}$ *Q*›
by *blast*
thus ‹*t* $\in \mathcal{D}$ *?rhs*›
proof *cases*
assume ‹*t-P* $\in \mathcal{D}$ *P* ∨ *t-Q* $\in \mathcal{D}$ *Q*›
with ‹*t-P* $\in \mathcal{T}$ *P*› ‹*t-Q* $\in \mathcal{T}$ *Q*› *setinter* ‹*ftF v*› ‹$t = u$ @ v› ‹*tF u*›
have ‹$t \in \mathcal{D}\ (P \llbracket A \rrbracket Q)$›
using *setinterleaving-sym* **by** (*simp add*: *D-Sync*) *blast*
thus ‹*t* $\in \mathcal{D}$ *?rhs*› **by** (*fact div*)
next
assume ‹*t-P* $\notin \mathcal{D}$ *P*› ‹*t-Q* $\notin \mathcal{D}$ *Q*›
with ‹*t-P* $\in \mathcal{T}$ *P*› ‹*t-Q* $\in \mathcal{T}$ *Q*› ‹$\boldsymbol{\alpha}(P) \subseteq A \vee \boldsymbol{\alpha}(Q) \subseteq A$›
have ‹$\{a.\ ev\ a \in set\ t\text{-}P\} \subseteq A \vee \{a.\ ev\ a \in set\ t\text{-}Q\} \subseteq A$›
by (*auto dest*: *subsetD intro*: *strict-events-of-memI*)
with *interleave-subsetL*[*OF* ‹*tF u*› - *setinter*]
interleave-subsetR[*OF* ‹*tF u*› - *setinter*]
have ‹$u = t\text{-}P \vee u = t\text{-}Q$› **by** *blast*
with ‹*length u* = *n*› **have** ‹*length t-P* = *n* ∨ *length t-Q* = *n*›
by *auto*
moreover from ‹*tF u*› *tickFree-interleave-iff*[*OF setinter*]

```
      have ‹tF t-P› ‹tF t-Q› by simp-all
      ultimately have ‹t-P ∈ 𝒟 (P ↓ n) ∨ t-Q ∈ 𝒟 (Q ↓ n)›
     using ‹t-P ∈ 𝒯 P› ‹t-Q ∈ 𝒯 Q› by (metis D-restriction-process_ptick I)
      moreover from ‹t-P ∈ 𝒯 P› ‹t-Q ∈ 𝒯 Q›
      have ‹t-P ∈ 𝒯 (P ↓ n)› ‹t-Q ∈ 𝒯 (Q ↓ n)›
        by (simp-all add: T-restriction-process_ptick I)
      ultimately show ‹t ∈ 𝒟 ?rhs›
        using ‹ftF v› ‹t = u @ v› ‹tF u› setinter
        by (simp add: D-Sync-optimized)
          (metis setinterleaving-sym)
    qed
  qed
 } note ∗ = this

 show ‹?rhs ⊑_FD ?lhs›
 proof (unfold refine-defs, safe)
  show ‹t ∈ 𝒟 ?lhs ⟹ t ∈ 𝒟 ?rhs› for t
  proof (elim D-restriction-process_ptick E)
   show ‹t ∈ 𝒟 (P ⟦A⟧ Q) ⟹ t ∈ 𝒟 ?rhs› by (fact div)
  next
   show ‹⟦t = u @ v; u ∈ 𝒯 (P ⟦A⟧ Q); length u = n; tF u; ftF v⟧
        ⟹ t ∈ 𝒟 ?rhs› for u v by (fact ∗)
  qed
 next
  show ‹(t, X) ∈ ℱ ?lhs ⟹ (t, X) ∈ ℱ ?rhs› for t X
  proof (elim F-restriction-process_ptick E)
   assume ‹(t, X) ∈ ℱ (P ⟦A⟧ Q)›
   then consider ‹t ∈ 𝒟 (P ⟦A⟧ Q)›
     | (fail) t-P t-Q X-P X-Q where ‹(t-P, X-P) ∈ ℱ P› ‹(t-Q,
X-Q) ∈ ℱ Q›
     ‹t setinterleaves ((t-P, t-Q), range tick ∪ ev ‘ A)›
     ‹X = (X-P ∪ X-Q) ∩ (range tick ∪ ev ‘ A) ∪ X-P ∩ X-Q›
    unfolding Sync-projs by blast
   thus ‹(t, X) ∈ ℱ ?rhs›
   proof cases
    from div D-F show ‹t ∈ 𝒟 (P ⟦A⟧ Q) ⟹ (t, X) ∈ ℱ ?rhs›
by blast
   next
    case fail
    thus ‹(t, X) ∈ ℱ ?rhs›
     by (auto simp add: F-Sync F-restriction-process_ptick)
   qed
  next
   show ‹⟦t = u @ v; u ∈ 𝒯 (P ⟦A⟧ Q); length u = n; tF u; ftF v⟧
        ⟹ (t, X) ∈ ℱ ?rhs› for u v by (simp add: ∗ is-processT8)
  qed
 qed
qed
```

corollary *restriction-process$_{ptick}$-MultiSync-FD* :
‹⟦A⟧ m ∈# M. P l ↓ n ⊑$_{FD}$ ⟦A⟧ m ∈# M. (P l ↓ n)›
proof (*induct M rule*: *induct-subset-mset-empty-single*)
case *1* **show** *?case* **by** *simp*
next
case (*2 m*) **show** *?case* **by** *simp*
next
case (*3 N m*)
show *?case*
by (*simp add*: ‹N ≠ {#}›)
(*fact trans-FD*[*OF restriction-process$_{ptick}$-Sync-FD mono-Sync-FD*[*OF idem-FD 3.hyps*(*4*)]])
qed

In the following corollary, we could be more precise by having the condition on at least *size M − 1* processes.

corollary *strict-events-of-subset-restriction-process$_{ptick}$-MultiSync* :
‹⟦A⟧ m ∈# M. P m ↓ n = (*if n = 0 then* ⊥ *else* ⟦A⟧ m ∈# M. (P m ↓ n))›
— *if n = 0 then* ⊥ *else* - is necessary because we can have M = {#}.
if ‹⋀m. m ∈# M ⟹ **α**(P m) ⊆ A›
proof (*split if-split*, *intro conjI impI*)
show ‹n = 0 ⟹ ⟦A⟧ m ∈# M. P m ↓ n = ⊥› **by** *simp*
next
show ‹⟦A⟧ m ∈# M. P m ↓ n = ⟦A⟧ m ∈# M. (P m ↓ n)› **if** ‹n ≠ 0›
proof (*induct M rule*: *induct-subset-mset-empty-single*)
case *1* **from** ‹n ≠ 0› **show** *?case* **by** *simp*
next
case (*2 m*) **show** *?case* **by** *simp*
next
case (*3 N m*)
have ‹(⟦A⟧ n∈#*add-mset m N*. P n) ↓ n = (P m ⟦A⟧ ⟦A⟧ n∈#N. P n) ↓ n›
by (*simp add*: ‹N ≠ {#}›)
also have ‹... = (P m ↓ n) ⟦A⟧ (⟦A⟧ n∈#N. P n ↓ n)›
by (*rule strict-events-of-subset-restriction-process$_{ptick}$-Sync*)
(*simp add*: *3.hyps*(*1*) ‹⋀m. m ∈# M ⟹ **α**(P m) ⊆ A›)
also have ‹(⟦A⟧ m∈#N. P m) ↓ n = ⟦A⟧ m∈#N. (P m ↓ n)› **by** (*fact 3.hyps*(*4*))
finally show *?case* **by** (*simp add*: ‹N ≠ {#}›)
qed
qed

corollary *restriction-process$_{ptick}$-Par* :
‹P || Q ↓ n = (P ↓ n) || (Q ↓ n)›

by (*simp add*: *strict-events-of-subset-restriction-process$_{ptick}$-Sync*)

corollary *restriction-process$_{ptick}$-MultiPar* :
‹|| $m \in\# M.\ P\ l \downarrow n$ = (*if* $n = 0$ *then* $\bot$ *else* || $m \in\# M.\ (P\ l \downarrow n)$)›
by (*simp add*: *strict-events-of-subset-restriction-process$_{ptick}$-MultiSync*)

6.3 Non Destructiveness

lemma *Sync-non-destructive* :
‹*non-destructive* ($\lambda(P, Q).\ P\ [\![A]\!]\ Q$)›
proof (*rule order-non-destructiveI*, *clarify*)
fix $P\ P'\ Q\ Q'$ :: ‹('a, 'r) *process$_{ptick}$*› **and** n
assume ‹$(P, Q) \downarrow n = (P', Q') \downarrow n$›
hence ‹$P \downarrow n = P' \downarrow n$› ‹$Q \downarrow n = Q' \downarrow n$›
by (*simp-all add*: *restriction-prod-def*)
show ‹$P\ [\![A]\!]\ Q \downarrow n \sqsubseteq_{FD} P'\ [\![A]\!]\ Q' \downarrow n$›
proof (*rule leFD-restriction-process$_{ptick}$I*)
show *div* : ‹$t \in \mathcal{D}\ (P'\ [\![A]\!]\ Q') \Longrightarrow t \in \mathcal{D}\ (P\ [\![A]\!]\ Q \downarrow n)$› **if** ‹*length* $t \leq n$› **for** t
proof (*unfold D-Sync-optimized*, *safe*)
fix $u\ v$ *t-P t-Q*
assume $* :$ ‹$t = u\ @\ v$› ‹*tF u*› ‹*ftF v*›
‹*u setinterleaves* ((*t-P*, *t-Q*), *range tick* $\cup$ *ev* ' A)›
‹*t-P* $\in \mathcal{D}\ P'$› ‹*t-Q* $\in \mathcal{T}\ Q'$›
from $*(1)$ ‹*length* $t \leq n$› **have** ‹*length* $u \leq n$› **by** *simp*
from ‹*length* $u \leq n$› *interleave-imp-lengthLR-le*[*OF* $*(4)$]
have ‹*length t-P* $\leq n$› ‹*length t-Q* $\leq n$› **by** *simp-all*
from ‹*t-Q* $\in \mathcal{T}\ Q'$› ‹*length t-Q* $\leq n$› ‹$Q \downarrow n = Q' \downarrow n$› **have** ‹*t-Q* $\in \mathcal{T}\ Q$›
by (*metis T-restriction-process$_{ptick}$I length-le-in-T-restriction-process$_{ptick}$*)
show ‹$u\ @\ v \in \mathcal{D}\ (P\ [\![A]\!]\ Q \downarrow n)$›
proof (*cases* ‹*length* $u = n$›)
assume ‹*length* $u = n$›
from ‹*t-P* $\in \mathcal{D}\ P'$› ‹*length t-P* $\leq n$› ‹$P \downarrow n = P' \downarrow n$› **have** ‹*t-P* $\in \mathcal{T}\ P$›
by (*simp add*: *D-T D-restriction-process$_{ptick}$I length-le-in-T-restriction-process$_{ptick}$*)
with ‹*t-Q* $\in \mathcal{T}\ Q$› $*(4)$ **have** ‹$u \in \mathcal{T}\ (P\ [\![A]\!]\ Q)$›
unfolding *T-Sync* **by** *blast*
with ‹*length* $u = n$› $*(2, 3)$ **show** ‹$u\ @\ v \in \mathcal{D}\ (P\ [\![A]\!]\ Q \downarrow n)$›
by (*simp add*: *D-restriction-process$_{ptick}$I is-processT7*)
next
assume ‹*length* $u \neq n$›
with ‹*length* $u \leq n$› **have** ‹*length* $u < n$› **by** *simp*
with *interleave-imp-lengthLR-le*[*OF* $*(4)$]
have ‹*length t-P* $< n$› **by** *simp*
with ‹*t-P* $\in \mathcal{D}\ P'$› ‹$P \downarrow n = P' \downarrow n$› **have** ‹*t-P* $\in \mathcal{D}\ P$›
by (*metis D-restriction-process$_{ptick}$I*
length-less-in-D-restriction-process$_{ptick}$)

```
      with ‹t-Q ∈ 𝒯 Q› *(2–4) have ‹u @ v ∈ 𝒟 (P ⟦A⟧ Q)›
        unfolding D-Sync by blast
      thus ‹u @ v ∈ 𝒟 (P ⟦A⟧ Q ↓ n)›
        by (simp add: D-restriction-process_ptick I)
    qed
  next
    fix u v t-P t-Q
    assume * : ‹t = u @ v› ‹tF u› ‹ftF v›
      ‹u setinterleaves ((t-Q, t-P), range tick ∪ ev ‘ A)›
      ‹t-P ∈ 𝒯 P'› ‹t-Q ∈ 𝒟 Q'›
    from *(1) ‹length t ≤ n› have ‹length u ≤ n› by simp
    from ‹length u ≤ n› interleave-imp-lengthLR-le[OF *(4)]
    have ‹length t-P ≤ n› ‹length t-Q ≤ n› by simp-all
    from ‹t-P ∈ 𝒯 P'› ‹length t-P ≤ n› ‹P ↓ n = P' ↓ n› have ‹t-P
∈ 𝒯 P›
    by (metis T-restriction-process_ptick I length-le-in-T-restriction-process_ptick)
    show ‹u @ v ∈ 𝒟 (P ⟦A⟧ Q ↓ n)›
    proof (cases ‹length u = n›)
      assume ‹length u = n›
      from ‹t-Q ∈ 𝒟 Q'› ‹length t-Q ≤ n› ‹Q ↓ n = Q' ↓ n› have
‹t-Q ∈ 𝒯 Q›
     by (simp add: D-T D-restriction-process_ptick I length-le-in-T-restriction-process_ptick)
      with ‹t-P ∈ 𝒯 P› *(4) have ‹u ∈ 𝒯 (P ⟦A⟧ Q)›
        unfolding T-Sync using setinterleaving-sym by blast
      with ‹length u = n› *(2, 3) show ‹u @ v ∈ 𝒟 (P ⟦A⟧ Q ↓ n)›
        by (simp add: D-restriction-process_ptick I is-processT7)
    next
      assume ‹length u ≠ n›
      with ‹length u ≤ n› have ‹length u < n› by simp
      with interleave-imp-lengthLR-le[OF *(4)]
      have ‹length t-Q < n› by simp
      with ‹t-Q ∈ 𝒟 Q'› ‹Q ↓ n = Q' ↓ n› have ‹t-Q ∈ 𝒟 Q›
        by (metis D-restriction-process_ptick I
            length-less-in-D-restriction-process_ptick)
      with ‹t-P ∈ 𝒯 P› *(2–4) have ‹u @ v ∈ 𝒟 (P ⟦A⟧ Q)›
        unfolding D-Sync by blast
      thus ‹u @ v ∈ 𝒟 (P ⟦A⟧ Q ↓ n)›
        by (simp add: D-restriction-process_ptick I)
    qed
  qed

  fix t X assume ‹(t, X) ∈ ℱ (P' ⟦A⟧ Q')› ‹length t ≤ n›
  then consider ‹t ∈ 𝒟 (P' ⟦A⟧ Q')›
    | (fail) t-P t-Q X-P X-Q
    where ‹(t-P, X-P) ∈ ℱ P'› ‹(t-Q, X-Q) ∈ ℱ Q'›
      ‹t setinterleaves ((t-P, t-Q), range tick ∪ ev ‘ A)›
      ‹X = (X-P ∪ X-Q) ∩ (range tick ∪ ev ‘ A) ∪ X-P ∩ X-Q›
    unfolding Sync-projs by blast
  thus ‹(t, X) ∈ ℱ (P ⟦A⟧ Q ↓ n)›
```

proof *cases*
from *div* ‹*length* $t \leq n$› *D-F*
show ‹$t \in \mathcal{D}\ (P' \llbracket A \rrbracket\ Q') \Longrightarrow (t,\ X) \in \mathcal{F}\ (P \llbracket A \rrbracket\ Q \downarrow n)$› **by** *blast*
next
case *fail*
show ‹$(t,\ X) \in \mathcal{F}\ (P \llbracket A \rrbracket\ Q \downarrow n)$›
proof (*cases* ‹*length* $t = n$›)
assume ‹*length* $t = n$›
from ‹*length* $t \leq n$› *interleave-imp-lengthLR-le*[*OF fail*(*3*)]
have ‹*length t-P* $\leq n$› ‹*length t-Q* $\leq n$› **by** *simp-all*
with *fail*(*1*, *2*) ‹$P \downarrow n = P' \downarrow n$› ‹$Q \downarrow n = Q' \downarrow n$›
have ‹*t-P* $\in \mathcal{T}\ P$› ‹*t-Q* $\in \mathcal{T}\ Q$›
by (*metis F-T T-restriction-process*$_{ptick}I$
length-le-in-T-restriction-process$_{ptick}$)+
from *F-imp-front-tickFree* ‹$(t,\ X) \in \mathcal{F}\ (P' \llbracket A \rrbracket\ Q')$›
have ‹*ftF t*› **by** *blast*
with *fail*(*3*) **consider** ‹*tF t*›
| *r s t-P' t-Q'* **where** ‹*t-P* $=$ *t-P'* @ [✓(*r*)]› ‹*t-Q* $=$ *t-Q'* @ [✓(*s*)]›
by (*metis F-imp-front-tickFree SyncWithTick-imp-NTF*
fail(*1*−*2*) *nonTickFree-n-frontTickFree*)
thus ‹$(t,\ X) \in \mathcal{F}\ (P \llbracket A \rrbracket\ Q \downarrow n)$›
proof *cases*
assume ‹*tF t*›
from ‹*t-P* $\in \mathcal{T}\ P$› ‹*t-Q* $\in \mathcal{T}\ Q$› *fail*(*3*)
have ‹$t \in \mathcal{T}\ (P \llbracket A \rrbracket\ Q)$› **unfolding** *T-Sync* **by** *blast*
hence ‹$t \in \mathcal{D}\ (P \llbracket A \rrbracket\ Q \downarrow n)$›
by (*simp add*: *D-restriction-process*$_{ptick}I$ ‹*length* $t = n$› ‹*tF t*›)
thus ‹$(t,\ X) \in \mathcal{F}\ (P \llbracket A \rrbracket\ Q \downarrow n)$› **by** (*simp add*: *is-processT8*)
next
fix *r s t-P' t-Q'* **assume** ‹*t-P* $=$ *t-P'* @ [✓(*r*)]› ‹*t-Q* $=$ *t-Q'* @ [✓(*s*)]›
have ‹(*t-P*, *X-P*) $\in \mathcal{F}\ P$›
by (*metis* ‹*t-P* $\in \mathcal{T}\ P$› ‹*t-P* $=$ *t-P'* @ [✓(*r*)]› *tick-T-F*)
moreover have ‹(*t-Q*, *X-Q*) $\in \mathcal{F}\ Q$›
by (*metis* ‹*t-Q* $\in \mathcal{T}\ Q$› ‹*t-Q* $=$ *t-Q'* @ [✓(*s*)]› *tick-T-F*)
ultimately have ‹$(t,\ X) \in \mathcal{F}\ (P \llbracket A \rrbracket\ Q)$›
using *fail*(*3*, *4*) **unfolding** *F-Sync* **by** *fast*
thus ‹$(t,\ X) \in \mathcal{F}\ (P \llbracket A \rrbracket\ Q \downarrow n)$›
by (*simp add*: *F-restriction-process*$_{ptick}I$)
qed
next
assume ‹*length* $t \neq n$›
with ‹*length* $t \leq n$› **have** ‹*length* $t < n$› **by** *simp*
with *interleave-imp-lengthLR-le*[*OF fail*(*3*)]
have ‹*length t-P* $< n$› ‹*length t-Q* $< n$› **by** *simp-all*
with *fail*(*1*, *2*) ‹$P \downarrow n = P' \downarrow n$› ‹$Q \downarrow n = Q' \downarrow n$›

have ‹(*t-P*, *X-P*) ∈ $\mathcal{F}$ *P*› ‹(*t-Q*, *X-Q*) ∈ $\mathcal{F}$ *Q*›
by (*metis F-restriction-process*$_{ptick}$*I*
length-less-in-F-restriction-process$_{ptick}$)+
with *fail*(*3*, *4*) **have** ‹(*t*, *X*) ∈ $\mathcal{F}$ (*P* ⟦*A*⟧ *Q*)›
unfolding *F-Sync* **by** *fast*
thus ‹(*t*, *X*) ∈ $\mathcal{F}$ (*P* ⟦*A*⟧ *Q* ↓ *n*)›
by (*simp add*: *F-restriction-process*$_{ptick}$*I*)
qed
qed
qed
qed

end

7 Non Destructiveness of Throw

7.1 Equality

lemma *Depth-Throw-1-is-constant*: ‹*P* Θ *a* ∈ *A*. *Q1 a* ↓ *1* = *P* Θ *a* ∈ *A*. *Q2 a* ↓ *1*›
proof (*rule FD-antisym*)
show ‹*P* Θ *a* ∈ *A*. *Q2 a* ↓ *1* ⊑$_{FD}$ *P* Θ *a* ∈ *A*. *Q1 a* ↓ *1*› **for** *Q1 Q2*
proof (*unfold refine-defs*, *safe*)
show *div* : ‹*t* ∈ $\mathcal{D}$ (*Throw P A Q1* ↓ *1*) ⟹ *t* ∈ $\mathcal{D}$ (*Throw P A Q2* ↓ *1*)› **for** *t*
proof (*elim D-restriction-process*$_{ptick}$*E*)
assume ‹*t* ∈ $\mathcal{D}$ (*P* Θ *a* ∈ *A*. *Q1 a*)› **and** ‹*length t* ≤ *1*›
from ‹*length t* ≤ *1*› **consider** ‹*t* = []› | *e* **where** ‹*t* = [*e*]› **by** (*cases t*) *simp-all*
thus ‹*t* ∈ $\mathcal{D}$ (*P* Θ *a* ∈ *A*. *Q2 a* ↓ *1*)›
proof *cases*
from ‹*t* ∈ $\mathcal{D}$ (*P* Θ *a* ∈ *A*. *Q1 a*)› **show** ‹*t* = [] ⟹ *t* ∈ $\mathcal{D}$ (*P* Θ *a* ∈ *A*. *Q2 a* ↓ *1*)›
by (*simp add*: *D-restriction-process*$_{ptick}$ *D-Throw*)
next
fix *e* **assume** ‹*t* = [*e*]›
with ‹*t* ∈ $\mathcal{D}$ (*P* Θ *a* ∈ *A*. *Q1 a*)›
consider ‹[] ∈ $\mathcal{D}$ *P*› | *a* **where** ‹*t* = [*ev a*]› ‹[*ev a*] ∈ $\mathcal{D}$ *P*› ‹*a* ∉ *A*›
| *a* **where** ‹*t* = [*ev a*]› ‹[*ev a*] ∈ $\mathcal{T}$ *P*› ‹*a* ∈ *A*›
by (*auto simp add*: *D-Throw disjoint-iff image-iff*)
(*metis D-T append-Nil event*$_{ptick}$*.exhaust process-charn*,
metis append-Nil empty-iff empty-set hd-append2 hd-in-set in-set-conv-decomp set-ConsD)
thus ‹*t* ∈ $\mathcal{D}$ (*P* Θ *a* ∈ *A*. *Q2 a* ↓ *1*)›

proof *cases*
show ‹$t = [ev\ a] \Longrightarrow [ev\ a] \in \mathcal{T}\ P \Longrightarrow a \in A \Longrightarrow t \in \mathcal{D}\ (P\ \Theta\ a \in A.\ Q2\ a \downarrow 1)$› **for** a
by (*simp add*: $D\text{-}restriction\text{-}process_{ptick}$ *T-Throw*)
(*metis append-Nil append-self-conv front-tickFree-charn inf-bot-left*
is-ev-def is-processT1-TR length-0-conv length-Cons list.set(*1*)
tickFree-Cons-iff tickFree-Nil)
next
show ‹$[] \in \mathcal{D}\ P \Longrightarrow t \in \mathcal{D}\ (P\ \Theta\ a \in A.\ Q2\ a \downarrow 1)$›
by (*simp flip*: *BOT-iff-Nil-D add*: *D-BOT*)
(*use* ‹$t = [e]$› *front-tickFree-single* **in** *blast*)
next
show ‹$t = [ev\ a] \Longrightarrow [ev\ a] \in \mathcal{D}\ P \Longrightarrow a \notin A \Longrightarrow t \in \mathcal{D}\ (P\ \Theta\ a \in A.\ Q2\ a \downarrow 1)$› **for** a
by (*simp add*: $D\text{-}restriction\text{-}process_{ptick}$ *D-Throw disjoint-iff image-iff*)
(*metis append.right-neutral empty-set* $event_{ptick}.disc(1)$ $event_{ptick}.sel(1)$
front-tickFree-Nil list.simps(*15*) *singletonD tickFree-Cons-iff tickFree-Nil*)
qed
qed
next
fix $u\ v$ **assume** $*$: ‹$t = u\ @\ v$› ‹$u \in \mathcal{T}\ (Throw\ P\ A\ Q1)$› ‹*length* $u = 1$› ‹$tF\ u$› ‹$ftF\ v$›
from ‹*length* $u = 1$› ‹$tF\ u$› **obtain** a **where** ‹$u = [ev\ a]$›
by (*cases* u) (*auto simp add*: *is-ev-def*)
with $*(2)$ **show** ‹$t \in \mathcal{D}\ (Throw\ P\ A\ Q2 \downarrow 1)$›
by (*simp add*: ‹$t = u\ @\ v$› $D\text{-}restriction\text{-}process_{ptick}$ *Throw-projs Cons-eq-append-conv*)
(*metis* (*no-types*) $*(3-5)$ *One-nat-def append-Nil empty-set inf-bot-left*
insert-disjoint(*2*) *is-processT1-TR list.simps*(*15*))
qed

show ‹$(t, X) \in \mathcal{F}\ (Throw\ P\ A\ Q1 \downarrow 1) \Longrightarrow (t, X) \in \mathcal{F}\ (Throw\ P\ A\ Q2 \downarrow 1)$› **for** $t\ X$
proof (*elim* $F\text{-}restriction\text{-}process_{ptick}E$)
assume ‹$(t, X) \in \mathcal{F}\ (P\ \Theta\ a \in A.\ Q1\ a)$› ‹*length* $t \leq 1$›
then consider ‹$t \in \mathcal{D}\ (P\ \Theta\ a \in A.\ Q1\ a)$› | ‹$(t, X) \in \mathcal{F}\ P$› ‹*set* $t \cap ev\ `\ A = \{\}$›
| a **where** ‹$t = [ev\ a]$› ‹$[ev\ a] \in \mathcal{T}\ P$› ‹$a \in A$›
by (*auto simp add*: *F-Throw D-Throw*)
thus ‹$(t, X) \in \mathcal{F}\ (P\ \Theta\ a \in A.\ Q2\ a \downarrow 1)$›
proof *cases*
from *D-F div* ‹*length* $t \leq 1$›
show ‹$t \in \mathcal{D}\ (P\ \Theta\ a \in A.\ Q1\ a) \Longrightarrow (t, X) \in \mathcal{F}\ (P\ \Theta\ a \in A.$

Q2 a ↓ 1)›
using *D-restriction-process*$_{ptick}$*I* **by** *blast*
next
show ‹(*t*, *X*) ∈ $\mathcal{F}$ *P* ⟹ *set t* ∩ *ev* ‘ *A* = {} ⟹ (*t*, *X*) ∈ $\mathcal{F}$ (*Throw P A Q2* ↓ *1*)›
by (*simp add*: *F-restriction-process*$_{ptick}$ *F-Throw*)
next
show ‹⟦*t* = [*ev a*]; [*ev a*] ∈ $\mathcal{T}$ *P*; *a* ∈ *A*⟧ ⟹ (*t*, *X*) ∈ $\mathcal{F}$ (*Throw P A Q2* ↓ *1*)› **for** *a*
by (*simp add*: *F-restriction-process*$_{ptick}$ *T-Throw*)
(*metis append.right-neutral append-Nil empty-set event*$_{ptick}$*.disc*(*1*) *front-tickFree-Nil inf-bot-left is-processT1-TR length-Cons list.size*(*3*) *tickFree-Cons-iff tickFree-Nil*)
qed
next
fix *u v* **assume** ∗ : ‹*t* = *u* @ *v*› ‹*u* ∈ $\mathcal{T}$ (*Throw P A Q1*)› ‹*length u* = *1*› ‹*tF u*› ‹*ftF v*›
from ‹*length u* = *1*› ‹*tF u*› **obtain** *a* **where** ‹*u* = [*ev a*]›
by (*cases u*) (*auto simp add*: *is-ev-def*)
with ∗(*2*) **show** ‹(*t*, *X*) ∈ $\mathcal{F}$ (*Throw P A Q2* ↓ *1*)›
by (*simp add*: ‹*t* = *u* @ *v*› *F-restriction-process*$_{ptick}$ *Throw-projs Cons-eq-append-conv*)
(*metis* (*no-types*) ∗(*3*−*5*) *One-nat-def append-Nil empty-set inf-bot-left insert-disjoint*(*2*) *is-processT1-TR list.simps*(*15*))
qed
qed

thus ‹*P* Θ *a* ∈ *A*. *Q2 a* ↓ *1* ⊑$_{FD}$ *P* Θ *a* ∈ *A*. *Q1 a* ↓ *1*› **by** *simp*
qed

7.2 Refinement

lemma *restriction-process*$_{ptick}$*-Throw-FD* :
‹(*P* Θ *a* ∈ *A*. *Q a*) ↓ *n* ⊑$_{FD}$ (*P* ↓ *n*) Θ *a* ∈ *A*. (*Q a* ↓ *n*)› (**is** ‹*?lhs* ⊑$_{FD}$ *?rhs*›)
proof (*unfold refine-defs*, *safe*)
show ‹*t* ∈ $\mathcal{D}$ *?lhs*› **if** ‹*t* ∈ $\mathcal{D}$ *?rhs*› **for** *t*
proof −
from ‹*t* ∈ $\mathcal{D}$ *?rhs*›
consider *t1 t2* **where** ‹*t* = *t1* @ *t2*› ‹*t1* ∈ $\mathcal{D}$ (*P* ↓ *n*)› ‹*tF t1*› ‹*set t1* ∩ *ev* ‘ *A* = {}› ‹*ftF t2*›
| *t1 a t2* **where** ‹*t* = *t1* @ *ev a* # *t2*› ‹*t1* @ [*ev a*] ∈ $\mathcal{T}$ (*P* ↓ *n*)› ‹*set t1* ∩ *ev* ‘ *A* = {}› ‹*a* ∈ *A*› ‹*t2* ∈ $\mathcal{D}$ (*Q a* ↓ *n*)›
unfolding *D-Throw* **by** *blast*
thus ‹*t* ∈ $\mathcal{D}$ *?lhs*›
proof *cases*
show ‹⟦*t* = *t1* @ *t2*; *t1* ∈ $\mathcal{D}$ (*P* ↓ *n*); *tF t1*; *set t1* ∩ *ev* ‘ *A* = {}; *ftF t2*⟧ ⟹ *t* ∈ $\mathcal{D}$ *?lhs*› **for** *t1 t2*

by (*elim D-restriction-process*$_{ptick}$*E*, *simp-all add*: *Throw-projs D-restriction-process*$_{ptick}$)
(*blast*, *metis* (*no-types*, *lifting*) *front-tickFree-append inf-sup-aci*(*8*) *inf-sup-distrib2 sup-bot-right*)
next
fix *t1 a t2*
assume ‹$t = t1 \ @ \ ev \ a \ \# \ t2$› ‹$t1 \ @ \ [ev \ a] \in \mathcal{T} \ (P \downarrow n)$›
‹*set t1* ∩ *ev* ‘ $A = \{\}$› ‹$a \in A$› ‹$t2 \in \mathcal{D} \ (Q \ a \downarrow n)$›
from ‹$t2 \in \mathcal{D} \ (Q \ a \downarrow n)$› **show** ‹$t \in \mathcal{D}$ *?lhs*›
proof (*elim D-restriction-process*$_{ptick}$*E*)
from ‹$t1 \ @ \ [ev \ a] \in \mathcal{T} \ (P \downarrow n)$› **show** ‹$t2 \in \mathcal{D} \ (Q \ a) \Longrightarrow$ *length* $t2 \leq n \Longrightarrow t \in \mathcal{D}$ *?lhs*›
proof (*elim T-restriction-process*$_{ptick}$*E*)
from ‹$a \in A$› ‹*set t1* ∩ *ev* ‘ $A = \{\}$› ‹$t = t1 \ @ \ ev \ a \ \# \ t2$›
show ‹$t2 \in \mathcal{D} \ (Q \ a) \Longrightarrow t1 \ @ \ [ev \ a] \in \mathcal{T} \ P \Longrightarrow t \in \mathcal{D}$ *?lhs*›
by (*auto simp add*: *D-restriction-process*$_{ptick}$ *D-Throw*)
next
fix *u v* **assume** ‹$t2 \in \mathcal{D} \ (Q \ a)$› ‹*length t2* ≤ *n*› ‹$t1 \ @ \ [ev \ a] = u \ @ \ v$›
‹$u \in \mathcal{T} \ P$› ‹*length u* = *n*› ‹*tF u*› ‹*ftF v*›
from ‹$t1 \ @ \ [ev \ a] = u \ @ \ v$› ‹*ftF v*› **consider** ‹$t1 \ @ \ [ev \ a] = u$›
| v' **where** ‹$t1 = u \ @ \ v'$› ‹$v = v' \ @ \ [ev \ a]$› ‹*ftF* v'›
by (*cases v rule*: *rev-cases*) (*simp-all add*: *front-tickFree-append-iff*)
thus ‹$t \in \mathcal{D}$ *?lhs*›
proof *cases*
from ‹$a \in A$› ‹*length u* = *n*› ‹*set t1* ∩ *ev* ‘ $A = \{\}$› ‹$t2 \in \mathcal{D} \ (Q \ a)$› ‹*tF u*› ‹$u \in \mathcal{T} \ P$›
show ‹$t1 \ @ \ [ev \ a] = u \Longrightarrow t \in \mathcal{D}$ *?lhs*›
by (*simp add*: *D-restriction-process*$_{ptick}$ *T-Throw* ‹$t = t1 \ @ \ ev \ a \ \# \ t2$›)
(*metis Cons-eq-appendI D-imp-front-tickFree append-Nil append-assoc is-processT1-TR*)
next
from ‹*ftF v*› ‹*length u* = *n*› ‹*set t1* ∩ *ev* ‘ $A = \{\}$› ‹$t = t1 \ @ \ ev \ a \ \# \ t2$› ‹$u \in \mathcal{T} \ P$›
show ‹$t1 = u \ @ \ v' \Longrightarrow v = v' \ @ \ [ev \ a] \Longrightarrow ftF \ v' \Longrightarrow t \in \mathcal{D}$ *?lhs*› **for** v'
by (*simp add*: *D-restriction-process*$_{ptick}$ *T-Throw*)
(*metis D-imp-front-tickFree Int-assoc Un-Int-eq*(*3*) *append-assoc*
front-tickFree-append front-tickFree-nonempty-append-imp
inf-bot-right list.distinct(*1*) *same-append-eq set-append* ‹$t \in \mathcal{D}$ *?rhs*›)
qed
qed
next
from ‹$t1 \ @ \ [ev \ a] \in \mathcal{T} \ (P \downarrow n)$›
show ‹⟦$t2 = u \ @ \ v$; $u \in \mathcal{T} \ (Q \ a)$; *length u* = *n*; *tF u*; *ftF v*⟧

$\Longrightarrow t \in \mathcal{D}$ *?lhs*› **for** *u v*
proof (*elim T-restriction-process*$_{ptick}E$)
assume ‹*t2* = *u* @ *v*› ‹*u* ∈ $\mathcal{T}$ (*Q a*)› ‹*length u* = *n*› ‹*tF u*›
‹*ftF v*› ‹*t1* @ [*ev a*] ∈ $\mathcal{T}$ *P*› ‹*length* (*t1* @ [*ev a*]) ≤ *n*›
from ‹*a* ∈ *A*› ‹*set t1* ∩ *ev* ' *A* = {}› ‹*t1* @ [*ev a*] ∈ $\mathcal{T}$ *P*› ‹*u* ∈ $\mathcal{T}$ (*Q a*)›
have ‹*t1* @ *ev a* # *u* ∈ $\mathcal{T}$ (*P* Θ *a*∈*A*. *Q a*)› **by** (*auto simp add*: *T-Throw*)
moreover have ‹*n* < *length* (*t1* @ *ev a* # *u*)› **by** (*simp add*: ‹*length u* = *n*›)
ultimately have ‹*t1* @ *ev a* # *u* ∈ $\mathcal{D}$ *?lhs*› **by** (*simp add*: *D-restriction-process*$_{ptick}I$)
moreover have ‹*t* = (*t1* @ *ev a* # *u*) @ *v*› **by** (*simp add*: ‹*t* = *t1* @ *ev a* # *t2*› ‹*t2* = *u* @ *v*›)
moreover from ‹*t1* @ [*ev a*] ∈ $\mathcal{T}$ *P*› ‹*tF u*› *append-T-imp-tickFree*
have ‹*tF* (*t1* @ *ev a* # *u*)› **by** *auto*
ultimately show ‹*t* ∈ $\mathcal{D}$ *?lhs*› **using** ‹*ftF v*› *is-processT7* **by** *blast*
next
fix *w x* **assume** ‹*t2* = *u* @ *v*› ‹*u* ∈ $\mathcal{T}$ (*Q a*)› ‹*length u* = *n*› ‹*tF u*› ‹*ftF v*›
‹*t1* @ [*ev a*] = *w* @ *x*› ‹*w* ∈ $\mathcal{T}$ *P*› ‹*length w* = *n*› ‹*tF w*› ‹*ftF x*›
from ‹*t1* @ [*ev a*] = *w* @ *x*› **consider** ‹*t1* @ [*ev a*] = *w*›
| *x'* **where** ‹*t1* = *w* @ *x'*› ‹*x* = *x'* @ [*ev a*]›
by (*cases x rule*: *rev-cases*) *simp-all*
thus ‹*t* ∈ $\mathcal{D}$ *?lhs*›
proof *cases*
assume ‹*t1* @ [*ev a*] = *w*›
with ‹*a* ∈ *A*› ‹*set t1* ∩ *ev* ' *A* = {}› ‹*u* ∈ $\mathcal{T}$ (*Q a*)› ‹*w* ∈ $\mathcal{T}$ *P*›
have ‹*t1* @ *ev a* # *u* ∈ $\mathcal{T}$ (*P* Θ *a*∈*A*. *Q a*)› **by** (*auto simp add*: *T-Throw*)
moreover have ‹*n* < *length* (*t1* @ *ev a* # *u*)› **by** (*simp add*: ‹*length u* = *n*›)
ultimately have ‹*t1* @ *ev a* # *u* ∈ $\mathcal{D}$ *?lhs*› **by** (*blast intro*: *D-restriction-process*$_{ptick}I$)
moreover have ‹*t* = (*t1* @ *ev a* # *u*) @ *v*›
by (*simp add*: ‹*t* = *t1* @ *ev a* # *t2*› ‹*t2* = *u* @ *v*›)
moreover from ‹*t1* @ [*ev a*] = *w*› ‹*tF u*› ‹*tF w*› **have** ‹*tF* (*t1* @ *ev a* # *u*)› **by** *auto*
ultimately show ‹*t* ∈ $\mathcal{D}$ *?lhs*› **using** ‹*ftF v*› *is-processT7* **by** *blast*
next
fix *x'* **assume** ‹*t1* = *w* @ *x'*› ‹*x* = *x'* @ [*ev a*]›
from ‹*set t1* ∩ *ev* ' *A* = {}› ‹*t1* = *w* @ *x'*› ‹*w* ∈ $\mathcal{T}$ *P*›
have ‹*w* ∈ $\mathcal{T}$ *P* ∧ *set w* ∩ *ev* ' *A* = {}› **by** *auto*
hence ‹*w* ∈ $\mathcal{T}$ (*P* Θ *a*∈*A*. *Q a*)› **by** (*simp add*: *T-Throw*)
with ‹*length w* = *n*› ‹*tF w*› **have** ‹*w* ∈ $\mathcal{D}$ *?lhs*›

by (*blast intro*: *D-restriction-process*$_{ptick}$*I*)
moreover have ‹*t* = *w* @ *x* @ *t2*›
by (*simp add*: ‹*t* = *t1* @ *ev a* # *t2*› ‹*t1* @ [*ev a*] = *w* @ *x*›)
moreover from *D-imp-front-tickFree*[*OF* ‹*t* ∈ 𝒟 *?rhs*›] ‹*t* = *t1* @ *ev a* # *t2*› ‹*t1* = *w* @ *x'*›
front-tickFree-nonempty-append-imp that tickFree-append-iff
have ‹*ftF* (*x* @ *t2*)› **by** (*simp add*: ‹*t* = *w* @ *x* @ *t2*› *front-tickFree-append-iff*)
ultimately show ‹*t* ∈ 𝒟 *?lhs*› **by** (*simp add*: ‹*tF w*› *is-processT7*)
qed
qed
qed
qed
qed

thus ‹(*t*, *X*) ∈ ℱ *?rhs* ⟹ (*t*, *X*) ∈ ℱ *?lhs*› **for** *t X*
by (*meson is-processT8 le-approx2 mono-Throw restriction-process*$_{ptick}$*-approx-self*)
qed

7.3 Non Destructiveness

lemma *Throw-non-destructive* :
‹*non-destructive* (λ(*P* :: ('*a*, '*r*) *process*$_{ptick}$, *Q*). *P* Θ *a* ∈ *A*. *Q a*)›
proof (*rule order-non-destructiveI*, *clarify*)
fix *P P'* :: ‹('*a*, '*r*) *process*$_{ptick}$› **and** *Q Q'* :: ‹'*a* ⇒ ('*a*, '*r*) *process*$_{ptick}$› **and** *n*
assume ‹(*P*, *Q*) ↓ *n* = (*P'*, *Q'*) ↓ *n*› ‹*0* < *n*›
hence ‹*P* ↓ *n* = *P'* ↓ *n*› ‹*Q* ↓ *n* = *Q'* ↓ *n*›
by (*simp-all add*: *restriction-prod-def*)

{ let *?lhs* = ‹*P* Θ *a* ∈ *A*. *Q a* ↓ *n*›
fix *t u v* **assume** ‹*t* = *u* @ *v*› ‹*u* ∈ 𝒯 (*Throw P' A Q'*)› ‹*length u* = *n*› ‹*tF u*› ‹*ftF v*›
from *this*(*2*) **consider** ‹*u* ∈ 𝒯 *P'*› ‹*set u* ∩ *ev* ‘ *A* = {}›
| (*divL*) *t1 t2* **where** ‹*u* = *t1* @ *t2*› ‹*t1* ∈ 𝒟 *P'*› ‹*tF t1*›
‹*set t1* ∩ *ev* ‘ *A* = {}› ‹*ftF t2*›
| (*traces*) *t1 a t2* **where** ‹*u* = *t1* @ *ev a* # *t2*› ‹*t1* @ [*ev a*] ∈ 𝒯 *P'*›
‹*set t1* ∩ *ev* ‘ *A* = {}› ‹*a* ∈ *A*› ‹*t2* ∈ 𝒯 (*Q' a*)›
unfolding *T-Throw* **by** *blast*
hence ‹*t* ∈ 𝒟 *?lhs*›
proof *cases*
assume ‹*u* ∈ 𝒯 *P'*› ‹*set u* ∩ *ev* ‘ *A* = {}›
from ‹*u* ∈ 𝒯 *P'*› ‹*length u* = *n*› ‹*P* ↓ *n* = *P'* ↓ *n*› **have** ‹*u* ∈ 𝒯 *P*›
by (*metis T-restriction-process*$_{ptick}$*I dual-order.refl*
length-le-in-T-restriction-process$_{ptick}$)

```
    with ‹set u ∩ ev ‘ A = {}› have ‹u ∈ 𝒯 (Throw P A Q)›
      by (simp add: T-Throw)
     with ‹ftF v› ‹t = u @ v› ‹tF u› ‹length u = n› show ‹t ∈ 𝒟
?lhs›
      by (simp add: D-restriction-process_ptick I is-processT7)
   next
    case divL
    show ‹t ∈ 𝒟 ?lhs›
    proof (cases ‹length t1 = n›)
     assume ‹length t1 = n›
     with ‹P ↓ n = P' ↓ n› divL(2,4) have ‹t1 ∈ 𝒯 (Throw P A
Q)›
       by (simp add: T-Throw)
        (metis D-T T-restriction-process_ptick I dual-order.refl
          length-le-in-T-restriction-process_ptick)
     with ‹ftF v› ‹length t1 = n› ‹t = u @ v› ‹tF u›
      divL(1,3,5) ‹length u = n› show ‹t ∈ 𝒟 ?lhs›
      by (auto simp add: D-restriction-process_ptick is-processT7)
    next
     assume ‹length t1 ≠ n›
     with ‹length u = n› divL(1) have ‹length t1 < n› by simp
     with ‹P ↓ n = P' ↓ n› divL(2,3,4) have ‹t1 ∈ 𝒟 P›
      by (simp add: D-Throw)
     (metis D-restriction-process_ptick I length-less-in-D-restriction-process_ptick)
     with ‹ftF v› ‹t = u @ v› divL(1,3,4) show ‹t ∈ 𝒟 ?lhs›
      by (simp add: D-restriction-process_ptick T-Throw)
       (metis ‹length u = n› ‹tF u› append.assoc divL(5))
    qed
   next
    case traces
    from ‹length u = n› traces(1)
    have ‹length (t1 @ [ev a]) ≤ n› ‹length t2 ≤ n› by simp-all
     from ‹P ↓ n = P' ↓ n› ‹t1 @ [ev a] ∈ 𝒯 P'› ‹length (t1 @ [ev
a]) ≤ n›
    have ‹t1 @ [ev a] ∈ 𝒯 P›
    by (metis T-restriction-process_ptick I length-le-in-T-restriction-process_ptick)
     moreover from ‹length t2 ≤ n› ‹t2 ∈ 𝒯 (Q' a)› ‹Q ↓ n = Q'
↓ n›
    have ‹t2 ∈ 𝒯 (Q a)›
      by (metis T-restriction-process_ptick I restriction-fun-def
          length-le-in-T-restriction-process_ptick)
    ultimately have ‹u ∈ 𝒯 (P Θ a ∈ A. Q a)›
      using traces(1,3,4) unfolding T-Throw by blast
    with ‹ftF v› ‹length u = n› ‹t = u @ v› ‹tF u›
   show ‹t ∈ 𝒟 ?lhs› by (auto simp add: D-restriction-process_ptick)
  qed
 } note * = this

 show ‹P Θ a ∈ A. Q a ↓ n ⊑_FD P' Θ a ∈ A. Q' a ↓ n› (is ‹?lhs
```

```
⊑FD ?rhs›)
  proof (unfold refine-defs, safe)
    show div : ‹t ∈ 𝒟 ?rhs ⟹ t ∈ 𝒟 ?lhs› for t
    proof (elim D-restriction-process_ptick E)
      assume ‹t ∈ 𝒟 (P' Θ a ∈ A. Q' a)› ‹length t ≤ n›
      from this(1) consider (divL) t1 t2 where ‹t = t1 @ t2› ‹t1 ∈
𝒟 P'›
        ‹tF t1› ‹set t1 ∩ ev ‘ A = {}› ‹ftF t2›
      | (divR) t1 a t2 where ‹t = t1 @ ev a # t2› ‹t1 @ [ev a] ∈ 𝒯
P'›
        ‹set t1 ∩ ev ‘ A = {}› ‹a ∈ A› ‹t2 ∈ 𝒟 (Q' a)›
        unfolding D-Throw by blast
      thus ‹t ∈ 𝒟 ?lhs›
      proof cases
        case divL
        show ‹t ∈ 𝒟 ?lhs›
        proof (cases ‹length t1 = n›)
          assume ‹length t1 = n›
          have ‹t1 ∈ 𝒯 P'› by (simp add: D-T divL(2))
          with ‹P ↓ n = P' ↓ n› ‹length t1 = n› have ‹t1 ∈ 𝒯 P›
            by (metis T-restriction-process_ptick I dual-order.eq-iff
                length-le-in-T-restriction-process_ptick)
          with divL(4) have ‹t1 ∈ 𝒯 (Throw P A Q)› by (simp add:
T-Throw)
          with ‹length t1 = n› divL(1,3,5) show ‹t ∈ 𝒟 ?lhs›
            by (simp add: D-restriction-process_ptick I is-processT7)
        next
          assume ‹length t1 ≠ n›
          with ‹length t ≤ n› divL(1) have ‹length t1 < n› by simp
          with ‹t1 ∈ 𝒟 P'› ‹P ↓ n = P' ↓ n› have ‹t1 ∈ 𝒟 P›
         by (metis D-restriction-process_ptick I length-less-in-D-restriction-process_ptick)
          with divL(3, 4) front-tickFree-Nil have ‹t1 ∈ 𝒟 (Throw P A
Q)›
            by (simp (no-asm) add: D-Throw) blast
          with divL(1,3,5) show ‹t ∈ 𝒟 ?lhs›
            by (simp add: D-restriction-process_ptick I is-processT7)
        qed
      next
        case divR
        from ‹length t ≤ n› divR(1)
        have ‹length (t1 @ [ev a]) ≤ n› ‹length t2 < n› by simp-all
        from ‹P ↓ n = P' ↓ n› ‹t1 @ [ev a] ∈ 𝒯 P'› ‹length (t1 @ [ev
a]) ≤ n›
        have ‹t1 @ [ev a] ∈ 𝒯 P›
       by (metis T-restriction-process_ptick I length-le-in-T-restriction-process_ptick)
        moreover from ‹length t2 < n› ‹t2 ∈ 𝒟 (Q' a)› ‹Q ↓ n = Q'
↓ n›
        have ‹t2 ∈ 𝒟 (Q a)›
          by (metis D-restriction-process_ptick I restriction-fun-def
```

```
        length-less-in-D-restriction-process_ptick)
      ultimately have ‹t ∈ 𝒟 (P Θ a ∈ A. Q a)›
       using divR(1,3,4) unfolding D-Throw by blast
      thus ‹t ∈ 𝒟 ?lhs› by (simp add: D-restriction-process_ptick I)
    qed
  next
    show ‹t = u @ v ⟹ u ∈ 𝒯 (Throw P' A Q') ⟹ length u = n
⟹
          tF u ⟹ ftF v ⟹ t ∈ 𝒟 ?lhs› for u v by (fact *)
  qed

  show ‹(t, X) ∈ ℱ ?rhs ⟹ (t, X) ∈ ℱ ?lhs› for t X
  proof (elim F-restriction-process_ptick E)
    assume ‹(t, X) ∈ ℱ (Throw P' A Q')› ‹length t ≤ n›
    from this(1) consider ‹t ∈ 𝒟 (Throw P' A Q')› | ‹(t, X) ∈ ℱ
P'› ‹set t ∩ ev ‘ A = {}›
      | (failR) t1 a t2 where ‹t = t1 @ ev a # t2› ‹t1 @ [ev a] ∈ 𝒯
P'›
        ‹set t1 ∩ ev ‘ A = {}› ‹a ∈ A› ‹(t2, X) ∈ ℱ (Q' a)›
      unfolding Throw-projs by auto
    thus ‹(t, X) ∈ ℱ ?lhs›
    proof cases
      assume ‹t ∈ 𝒟 (Throw P' A Q')›
      hence ‹t ∈ 𝒟 ?rhs› by (simp add: D-restriction-process_ptick I)
      with D-F div show ‹(t, X) ∈ ℱ ?lhs› by blast
    next
      assume ‹(t, X) ∈ ℱ P'› ‹set t ∩ ev ‘ A = {}›
      from ‹(t, X) ∈ ℱ P'› ‹P ↓ n = P' ↓ n› ‹length t ≤ n› have
‹t ∈ 𝒯 P›
       by (metis F-T T-restriction-process_ptick I length-le-in-T-restriction-process_ptick)
       with ‹set t ∩ ev ‘ A = {}› have ‹t ∈ 𝒯 (Throw P A Q)› by
(simp add: T-Throw)
        from F-imp-front-tickFree ‹(t, X) ∈ ℱ P'› have ‹ftF t› by
blast
       thus ‹(t, X) ∈ ℱ ?lhs›
       proof (elim front-tickFreeE)
         show ‹(t, X) ∈ ℱ ?lhs› if ‹tF t›
         proof (cases ‹length t = n›)
           assume ‹length t = n›
            with ‹t ∈ 𝒯 (Throw P A Q)› ‹tF t› front-tickFree-charn
show ‹(t, X) ∈ ℱ ?lhs›
             by (simp add: F-restriction-process_ptick) blast
         next
           assume ‹length t ≠ n›
           with ‹length t ≤ n› have ‹length t < n› by simp
           with ‹(t, X) ∈ ℱ P'› ‹P ↓ n = P' ↓ n› have ‹(t, X) ∈ ℱ
P›
         by (simp add: F-restriction-process_ptick length-less-in-F-restriction-process_ptick)
           with ‹set t ∩ ev ‘ A = {}› have ‹(t, X) ∈ ℱ (Throw P A
```

Q)› **by** (*simp add*: *F-Throw*)
thus ‹$(t, X) \in \mathcal{F}$ *?lhs*› **by** (*simp add*: *F-restriction-process*$_{ptick}I$)
qed
next
fix t' r **assume** ‹$t = t'$ @ [✓(r)]›
with ‹$t \in \mathcal{T}$ (*Throw P A Q*)› **have** ‹$(t, X) \in \mathcal{F}$ (*Throw P A Q*)›
by (*simp add*: *tick-T-F*)
thus ‹$(t, X) \in \mathcal{F}$ *?lhs*› **by** (*simp add*: *F-restriction-process*$_{ptick}I$)
qed
next
case *failR*
from ‹*length* $t \leq n$› *failR*(*1*)
have ‹*length* (*t1* @ [*ev a*]) $\leq n$› ‹*length t2* $< n$› **by** *simp-all*
from ‹$P \downarrow n = P' \downarrow n$› ‹*t1* @ [*ev a*] $\in \mathcal{T}$ P'› ‹*length* (*t1* @ [*ev a*]) $\leq n$›
have ‹*t1* @ [*ev a*] $\in \mathcal{T}$ P›
by (*metis T-restriction-process*$_{ptick}I$ *length-le-in-T-restriction-process*$_{ptick}$)
moreover from ‹*length t2* $< n$› ‹$(t2, X) \in \mathcal{F}$ (Q' a)› ‹$Q \downarrow n = Q' \downarrow n$›
have ‹$(t2, X) \in \mathcal{F}$ (Q a)›
by (*metis F-restriction-process*$_{ptick}I$ *restriction-fun-def*
length-less-in-F-restriction-process$_{ptick}$)
ultimately have ‹$(t, X) \in \mathcal{F}$ (P Θ $a \in A$. Q a)›
using *failR*(*1*,*3*,*4*) **unfolding** *F-Throw* **by** *blast*
thus ‹$(t, X) \in \mathcal{F}$ *?lhs*› **by** (*simp add*: *F-restriction-process*$_{ptick}I$)
qed
next
show ‹$t = u$ @ $v \Longrightarrow u \in \mathcal{T}$ (*Throw P' A Q'*) $\Longrightarrow$ *length* $u = n$ $\Longrightarrow$
tF $u \Longrightarrow$ *ftF* $v \Longrightarrow (t, X) \in \mathcal{F}$ *?lhs*› **for** u v
by (*simp add*: $*$ *is-processT8*)
qed
qed
qed

lemma *ThrowR-constructive-if-disjoint-initials* :
‹*constructive* (λQ :: $'a \Rightarrow$ ($'a$, $'r$) *process*$_{ptick}$. P Θ $a \in A$. Q a)›
if ‹$A \cap \{e.\ ev\ e \in P^0\} = \{\}$›
proof (*rule order-constructiveI*)
fix Q Q' :: ‹$'a \Rightarrow$ ($'a$, $'r$) *process*$_{ptick}$› **and** n **assume** ‹$Q \downarrow n = Q' \downarrow n$›

{ **let** *?lhs* = ‹*Throw P A Q* $\downarrow$ *Suc n*›
fix t u v
assume ‹$t = u$ @ v› ‹$u \in \mathcal{T}$ (*Throw P A Q'*)› ‹*length* u = *Suc n*› ‹*tF* u› ‹*ftF* v›

from ‹$u \in \mathcal{T}$ (*Throw P A Q'*)› **consider** ‹$u \in \mathcal{T}$ *P*› ‹*set u* ∩ *ev* ‘ *A* = {}›
| (*divL*) *t1 t2* **where** ‹u = *t1* @ *t2*› ‹*t1* ∈ $\mathcal{D}$ *P*› ‹*tF t1*›
‹*set t1* ∩ *ev* ‘ *A* = {}› ‹*ftF t2*›
| (*traces*) *t1 a t2* **where** ‹u = *t1* @ *ev a* # *t2*› ‹*t1* @ [*ev a*] ∈ $\mathcal{T}$ *P*›
‹*set t1* ∩ *ev* ‘ *A* = {}› ‹*a* ∈ *A*› ‹*t2* ∈ $\mathcal{T}$ (*Q' a*)›
unfolding *T-Throw* **by** *blast*
hence ‹*u* ∈ $\mathcal{D}$ *?lhs*›
proof *cases*
assume ‹*u* ∈ $\mathcal{T}$ *P*› ‹*set u* ∩ *ev* ‘ *A* = {}›
hence ‹*u* ∈ $\mathcal{T}$ (*Throw P A Q*)› **by** (*simp add*: *T-Throw*)
with ‹*length u* = *Suc n*› ‹*tF u*› **show** ‹*u* ∈ $\mathcal{D}$ *?lhs*›
by (*simp add*: *D-restriction-process*$_{ptick}$*I*)
next
case *divL*
hence ‹*u* ∈ $\mathcal{D}$ (*Throw P A Q*)› **by** (*auto simp add*: *D-Throw*)
thus ‹*u* ∈ $\mathcal{D}$ *?lhs*› **by** (*simp add*: *D-restriction-process*$_{ptick}$*I*)
next
case *traces*
from ‹*length u* = *Suc n*› *traces*(*1*) **have** ‹*length t2* ≤ *n*› **by** *simp*
with ‹*t2* ∈ $\mathcal{T}$ (*Q' a*)› ‹*Q* ↓ *n* = *Q'* ↓ *n*› **have** ‹*t2* ∈ $\mathcal{T}$ (*Q a*)›
by (*metis T-restriction-process*$_{ptick}$*I restriction-fun-def*
length-le-in-T-restriction-process$_{ptick}$)
with *traces*(*1*−*4*) **have** ‹*u* ∈ $\mathcal{T}$ (*Throw P A Q*)› **by** (*auto simp add*: *T-Throw*)
thus ‹*u* ∈ $\mathcal{D}$ *?lhs*›
by (*simp add*: *D-restriction-process*$_{ptick}$*I* ‹*length u* = *Suc n*› ‹*tF u*›)
qed
hence ‹*t* ∈ $\mathcal{D}$ *?lhs*› **by** (*simp add*: ‹*ftF v*› ‹*t* = *u* @ *v*› ‹*tF u*› *is-processT7*)
} note ∗ = *this*

show ‹(*P* Θ *a* ∈ *A*. *Q a*) ↓ *Suc n* $\sqsubseteq_{FD}$ *P* Θ *a* ∈ *A*. *Q' a* ↓ *Suc n*› (**is** ‹*?lhs* $\sqsubseteq_{FD}$ *?rhs*›)
proof (*unfold refine-defs*, *safe*)
show *div* : ‹*t* ∈ $\mathcal{D}$ *?rhs* ⟹ *t* ∈ $\mathcal{D}$ *?lhs*› **for** *t*
proof (*elim D-restriction-process*$_{ptick}$*E*)
assume ‹*t* ∈ $\mathcal{D}$ (*P* Θ *a* ∈ *A*. *Q' a*)› ‹*length t* ≤ *Suc n*›
from *this*(*1*) **consider** (*divL*) *t1 t2* **where** ‹*t* = *t1* @ *t2*› ‹*t1* ∈ $\mathcal{D}$ *P*›
‹*tF t1*› ‹*set t1* ∩ *ev* ‘ *A* = {}› ‹*ftF t2*›
| (*divR*) *t1 a t2* **where** ‹*t* = *t1* @ *ev a* # *t2*› ‹*t1* @ [*ev a*] ∈ $\mathcal{T}$ *P*›
‹*set t1* ∩ *ev* ‘ *A* = {}› ‹*a* ∈ *A*› ‹*t2* ∈ $\mathcal{D}$ (*Q' a*)›
unfolding *D-Throw* **by** *blast*
thus ‹*t* ∈ $\mathcal{D}$ *?lhs*›
proof *cases*

case *divL*
hence ‹$t \in \mathcal{D}\ (P\ \Theta\ a \in A.\ Q\ a)$› **by** (*auto simp add*: *D-Throw*)
thus ‹$t \in \mathcal{D}$ *?lhs*› **by** (*simp add*: *D-restriction-process*$_{ptick}I$)
next
case *divR*
from *divR*(*2*,*4*) *that* **have** ‹*t1* $\neq$ []›
by (*cases t1*) (*auto intro*: *initials-memI*)
with *divR*(*1*) ‹*length t* $\leq$ *Suc n*› *nat-less-le* **have** ‹*length t2* $<$ *n*› **by** *force*
with ‹*t2* $\in \mathcal{D}\ (Q'\ a)$› ‹$Q \downarrow n = Q' \downarrow n$› **have** ‹*t2* $\in \mathcal{D}\ (Q\ a)$›
by (*metis D-restriction-process*$_{ptick}I$ *restriction-fun-def*
length-less-in-D-restriction-process$_{ptick}$)
with *divR*(*1*−*4*) **have** ‹$t \in \mathcal{D}\ (Throw\ P\ A\ Q)$› **by** (*auto simp add*: *D-Throw*)
thus ‹$t \in \mathcal{D}$ *?lhs*› **by** (*simp add*: *D-restriction-process*$_{ptick}I$)
qed
next
show ‹$t = u\ @\ v \Longrightarrow u \in \mathcal{T}\ (Throw\ P\ A\ Q') \Longrightarrow length\ u = Suc\ n \Longrightarrow$
$tF\ u \Longrightarrow ftF\ v \Longrightarrow t \in \mathcal{D}$ *?lhs*› **for** *u v* **by** (*fact* $*$)
qed

show ‹$(t, X) \in \mathcal{F}$ *?rhs* $\Longrightarrow (t, X) \in \mathcal{F}$ *?lhs*› **for** *t X*
proof (*elim F-restriction-process*$_{ptick}E$)
assume ‹$(t, X) \in \mathcal{F}\ (Throw\ P\ A\ Q')$› ‹*length t* $\leq$ *Suc n*›
from *this*(*1*) **consider** ‹$t \in \mathcal{D}\ (Throw\ P\ A\ Q')$› | ‹$(t, X) \in \mathcal{F}\ P$› ‹*set t* $\cap$ *ev* ‘ *A* = {}›
| (*failR*) *t1 a t2* **where** ‹$t = t1\ @\ ev\ a\ \#\ t2$› ‹$t1\ @\ [ev\ a] \in \mathcal{T}\ P$›
‹*set t1* $\cap$ *ev* ‘ *A* = {}› ‹*a* $\in$ *A*› ‹$(t2, X) \in \mathcal{F}\ (Q'\ a)$›
unfolding *Throw-projs* **by** *auto*
thus ‹$(t, X) \in \mathcal{F}$ *?lhs*›
proof *cases*
assume ‹$t \in \mathcal{D}\ (Throw\ P\ A\ Q')$›
hence ‹$t \in \mathcal{D}$ *?rhs*› **by** (*simp add*: *D-restriction-process*$_{ptick}I$)
with *D-F div* **show** ‹$(t, X) \in \mathcal{F}$ *?lhs*› **by** *blast*
next
assume ‹$(t, X) \in \mathcal{F}\ P$› ‹*set t* $\cap$ *ev* ‘ *A* = {}›
hence ‹$(t, X) \in \mathcal{F}\ (Throw\ P\ A\ Q)$› **by** (*simp add*: *F-Throw*)
thus ‹$(t, X) \in \mathcal{F}$ *?lhs*› **by** (*simp add*: *F-restriction-process*$_{ptick}I$)
next
case *failR*
from *failR*(*2*, *4*) *that* **have** ‹*t1* $\neq$ []›
by (*cases t1*) (*auto intro*: *initials-memI*)
with *failR*(*1*) ‹*length t* $\leq$ *Suc n*› *nat-less-le* **have** ‹*length t2* $<$ *n*› **by** *force*
with ‹$(t2, X) \in \mathcal{F}\ (Q'\ a)$› ‹$Q \downarrow n = Q' \downarrow n$› **have** ‹$(t2, X) \in \mathcal{F}\ (Q\ a)$›
by (*metis F-restriction-process*$_{ptick}I$ *restriction-fun-def*

length-less-in-F-restriction-process$_{ptick}$)
with *failR*(*1*−*4*) **have** ‹$(t,\ X) \in \mathcal{F}\ (Throw\ P\ A\ Q)$› **by** (*auto simp add*: *F-Throw*)
thus ‹$(t,\ X) \in \mathcal{F}$ *?lhs*› **by** (*simp add*: *F-restriction-process*$_{ptick}I$)
qed
next
show ‹$t = u\ @\ v \Longrightarrow u \in \mathcal{T}\ (Throw\ P\ A\ Q') \Longrightarrow length\ u = Suc\ n \Longrightarrow$
tF u $\Longrightarrow$ *ftF v* $\Longrightarrow$ $(t,\ X) \in \mathcal{F}$ *?lhs*› **for** *u v*
by (*simp add*: ∗ *is-processT8*)
qed
qed
qed

8 Non Destructiveness of Interrupt

8.1 Refinement

lemma *restriction-process*$_{ptick}$*-Interrupt-FD* :
‹$P \triangle Q \downarrow n \sqsubseteq_{FD} (P \downarrow n) \triangle (Q \downarrow n)$› (**is** ‹*?lhs* $\sqsubseteq_{FD}$ *?rhs*›)
proof (*unfold refine-defs*, *safe*)
show ∗ : ‹$t \in \mathcal{D}$ *?lhs*› **if** ‹$t \in \mathcal{D}$ *?rhs*› **for** *t*
proof −
from ‹$t \in \mathcal{D}$ *?rhs*› **consider** ‹$t \in \mathcal{D}\ (P \downarrow n)$›
| *u v* **where** ‹$t = u\ @\ v$› ‹$u \in \mathcal{T}\ (P \downarrow n)$› ‹*tF u*› ‹$v \in \mathcal{D}\ (Q \downarrow n)$›
by (*simp add*: *D-Interrupt*) *blast*
thus ‹$t \in \mathcal{D}$ *?lhs*›
proof *cases*
show ‹$t \in \mathcal{D}\ (P \downarrow n) \Longrightarrow t \in \mathcal{D}$ *?lhs*›
by (*elim D-restriction-process*$_{ptick}E$) (*auto simp add*: *D-restriction-process*$_{ptick}$ *Interrupt-projs*)
next
fix *u v* **assume** ‹$t = u\ @\ v$› ‹$u \in \mathcal{T}\ (P \downarrow n)$› ‹*tF u*› ‹$v \in \mathcal{D}\ (Q \downarrow n)$›
from ‹$v \in \mathcal{D}\ (Q \downarrow n)$› **show** ‹$t \in \mathcal{D}$ *?lhs*›
proof (*elim D-restriction-process*$_{ptick}E$)
from ‹$t = u\ @\ v$› ‹$u \in \mathcal{T}\ (P \downarrow n)$› ‹*tF u*› **show** ‹$v \in \mathcal{D}\ Q \Longrightarrow t \in \mathcal{D}$ *?lhs*›
by (*auto simp add*: *restriction-process*$_{ptick}$*-projs Interrupt-projs D-imp-front-tickFree front-tickFree-append*)
next
fix *w x* **assume** ‹$v = w\ @\ x$› ‹$w \in \mathcal{T}\ Q$› ‹*length w* = *n*› ‹*tF w*› ‹*ftF x*›
from ‹$u \in \mathcal{T}\ (P \downarrow n)$› **consider** ‹$u \in \mathcal{D}\ (P \downarrow n)$› | ‹$u \in \mathcal{T}\ P$› ‹*length u* ≤ *n*›
by (*elim T-restriction-process*$_{ptick}E$) (*auto simp add*: *D-restriction-process*$_{ptick}$)
thus ‹$t \in \mathcal{D}$ *?lhs*›

proof *cases*
assume ‹$u \in \mathcal{D}\ (P \downarrow n)$›
with *D-imp-front-tickFree* ‹$t = u \mathbin{@} v$› ‹$tF\ u$› ‹$v \in \mathcal{D}\ (Q \downarrow n)$› *is-processT7*
have ‹$t \in \mathcal{D}\ (P \downarrow n)$› **by** *blast*
thus ‹$t \in \mathcal{D}$ *?lhs*› **by** (*elim D-restriction-process*$_{ptick}$*E*)
(*auto simp add*: *D-restriction-process*$_{ptick}$ *Interrupt-projs*)
next
assume ‹$u \in \mathcal{T}\ P$› ‹*length* $u \leq n$›
hence ‹$t = take\ n\ (u \mathbin{@} w) \mathbin{@} drop\ (n - length\ u)\ w \mathbin{@} x \wedge$
$take\ n\ (u \mathbin{@} w) \in \mathcal{T}\ (P \mathbin{\triangle} Q) \wedge length\ (take\ n\ (u \mathbin{@} w)) = n \wedge$
$tF\ (take\ n\ (u \mathbin{@} w)) \wedge ftF\ (drop\ (n - length\ u)\ w \mathbin{@} x)$›
by (*simp add*: ‹$t = u \mathbin{@} v$› ‹$v = w \mathbin{@} x$› ‹*length* $w = n$› ‹$tF\ u$› *T-Interrupt*)
(*metis* ‹$ftF\ x$› ‹$tF\ u$› ‹$tF\ w$› ‹$w \in \mathcal{T}\ Q$› *append-take-drop-id front-tickFree-append is-processT3-TR-append tickFree-append-iff*)
with *D-restriction-process*$_{ptick}$ **show** ‹$t \in \mathcal{D}$ *?lhs*› **by** *blast*
qed
qed
qed
qed

show ‹$(t, X) \in \mathcal{F}$ *?rhs* $\Longrightarrow (t, X) \in \mathcal{F}$ *?lhs*› **for** *t X*
by (*meson* ∗ *is-processT8 mono-Interrupt proc-ord2a restriction-process*$_{ptick}$*-approx-self*)
qed

8.2 Non Destructiveness

lemma *Interrupt-non-destructive* :
‹*non-destructive* (λ(P :: ('*a*, '*r*) *process*$_{ptick}$, Q). $P \mathbin{\triangle} Q$)›
proof (*rule order-non-destructiveI*, *clarify*)
fix $P\ P'\ Q\ Q'$ :: ‹('*a*, '*r*) *process*$_{ptick}$› **and** *n*
assume ‹$(P, Q) \downarrow n = (P', Q') \downarrow n$› ‹$0 < n$›
hence ‹$P \downarrow n = P' \downarrow n$› ‹$Q \downarrow n = Q' \downarrow n$›
by (*simp-all add*: *restriction-prod-def*)

{ let *?lhs* = ‹$P \mathbin{\triangle} Q \downarrow n$›
fix $t\ u\ v$ **assume** ‹$t = u \mathbin{@} v$› ‹$u \in \mathcal{T}\ (P' \mathbin{\triangle} Q')$› ‹*length* $u = n$› ‹$tF\ u$› ‹$ftF\ v$›
from *this*(*2*) ‹$tF\ u$› **obtain** *u1 u2*
where ‹$u = u1 \mathbin{@} u2$› ‹$u1 \in \mathcal{T}\ P'$› ‹$tF\ u1$› ‹$u2 \in \mathcal{T}\ Q'$› ‹$tF\ u2$›
by (*simp add*: *T-Interrupt*)
(*metis append-Nil2 is-processT1-TR tickFree-append-iff*)

from ‹*length* $u = n$› ‹$u = u1 \mathbin{@} u2$›
have ‹*length* $u1 \leq n$› ‹*length* $u2 \leq n$› **by** *simp-all*
with ‹$u1 \in \mathcal{T}\ P'$› ‹$P \downarrow n = P' \downarrow n$› ‹$u2 \in \mathcal{T}\ Q'$› ‹$Q \downarrow n = Q' \downarrow$

```
n›
    have ‹u1 ∈ 𝒯 P› ‹u2 ∈ 𝒯 Q›
      by (metis T-restriction-process_ptick I
          length-le-in-T-restriction-process_ptick)+
    with ‹tF u1› have ‹u ∈ 𝒯 (P △ Q)›
      by (auto simp add: ‹u = u1 @ u2› T-Interrupt)
    with ‹length u = n› ‹tF u› have ‹u ∈ 𝒟 ?lhs›
      by (simp add: D-restriction-process_ptick I)
     hence ‹t ∈ 𝒟 ?lhs› by (simp add: ‹ftF v› ‹t = u @ v› ‹tF u›
is-processT7)
  } note ∗ = this

  show ‹P △ Q ↓ n ⊑_FD P' △ Q' ↓ n› (is ‹?lhs ⊑_FD ?rhs›)
  proof (unfold refine-defs, safe)
    show div : ‹t ∈ 𝒟 ?rhs ⟹ t ∈ 𝒟 ?lhs› for t
    proof (elim D-restriction-process_ptick E)
      assume ‹t ∈ 𝒟 (P' △ Q')› ‹length t ≤ n›
      from this(1) consider ‹t ∈ 𝒟 P'›
        | (divR) t1 t2 where ‹t = t1 @ t2› ‹t1 ∈ 𝒯 P'› ‹tF t1› ‹t2 ∈
𝒟 Q'›
        unfolding D-Interrupt by blast
      thus ‹t ∈ 𝒟 ?lhs›
      proof cases
        assume ‹t ∈ 𝒟 P'›
        hence ‹ftF t› by (simp add: D-imp-front-tickFree)
        thus ‹t ∈ 𝒟 ?lhs›
        proof (elim front-tickFreeE)
          show ‹t ∈ 𝒟 ?lhs› if ‹tF t›
          proof (cases ‹length t = n›)
            assume ‹length t = n›
            from ‹P ↓ n = P' ↓ n› ‹t ∈ 𝒟 P'› ‹length t ≤ n› have ‹t
∈ 𝒯 P›
              by (metis D-T T-restriction-process_ptick I
                  length-le-in-T-restriction-process_ptick)
            with ‹tF t› ‹length t = n› front-tickFree-Nil show ‹t ∈ 𝒟
?lhs›
                    by (simp (no-asm) add: D-restriction-process_ptick
T-Interrupt) blast
          next
            assume ‹length t ≠ n›
            with ‹length t ≤ n› have ‹length t < n› by simp
            with ‹P ↓ n = P' ↓ n› ‹t ∈ 𝒟 P'› have ‹t ∈ 𝒟 P›
              by (metis D-restriction-process_ptick I
                  length-less-in-D-restriction-process_ptick)
            thus ‹t ∈ 𝒟 ?lhs› by (simp add: D-restriction-process_ptick I
D-Interrupt)
          qed
        next
          fix t' r assume ‹t = t' @ [✓(r)]› ‹tF t'›
```

with ‹$t \in \mathcal{D}\ P'$› ‹*length* $t \leq n$›
have ‹$t' \in \mathcal{D}\ P'$› ‹*length* $t' < n$› **by** (*auto intro*: *is-processT9*)
with ‹$P \downarrow n = P' \downarrow n$› **have** ‹$t' \in \mathcal{D}\ P$›
by (*metis D-restriction-process*$_{ptick}I$
length-less-in-D-restriction-process$_{ptick}$)
with ‹$t = t' @ [\checkmark(r)]$› ‹$tF\ t'$› **have** ‹$t \in \mathcal{D}\ P$› **by** (*simp add*: *is-processT7*)
thus ‹$t \in \mathcal{D}$ *?lhs*› **by** (*simp add*: *D-restriction-process*$_{ptick}I$ *D-Interrupt*)
qed
next
fix $u\ v$ **assume** ‹$t = u @ v$› ‹$u \in \mathcal{T}\ P'$› ‹$tF\ u$› ‹$v \in \mathcal{D}\ Q'$›
from ‹$t = u @ v$› ‹*length* $t \leq n$› **have** ‹*length* $u \leq n$› **by** *simp*
with ‹$P \downarrow n = P' \downarrow n$› ‹$u \in \mathcal{T}\ P'$› **have** ‹$u \in \mathcal{T}\ P$›
by (*metis T-restriction-process*$_{ptick}I$ *length-le-in-T-restriction-process*$_{ptick}$)
show ‹$t \in \mathcal{D}$ *?lhs*›
proof (*cases* ‹*length* $v = n$›)
assume ‹*length* $v = n$›
with ‹$t = u @ v$› ‹*length* $t \leq n$› **have** ‹$u = []$› **by** *simp*
from *D-imp-front-tickFree* ‹$v \in \mathcal{D}\ Q'$› **have** ‹$ftF\ v$› **by** *blast*
thus ‹$t \in \mathcal{D}$ *?lhs*›
proof (*elim front-tickFreeE*)
assume ‹$tF\ v$›
from ‹*length* $v = n$› ‹$Q \downarrow n = Q' \downarrow n$› ‹$v \in \mathcal{D}\ Q'$› **have** ‹$v \in \mathcal{T}\ Q$›
by (*metis D-T D-restriction-process*$_{ptick}I$ *dual-order.refl*
length-le-in-T-restriction-process$_{ptick}$)
with ‹$tF\ u$› ‹$u \in \mathcal{T}\ P$› **have** ‹$t \in \mathcal{T}\ (P \triangle Q)$›
by (*auto simp add*: ‹$t = u @ v$› *T-Interrupt*)
with ‹*length* $v = n$› ‹$t = u @ v$› ‹$tF\ v$› ‹$u = []$› **show** ‹$t \in \mathcal{D}$ *?lhs*›
by (*simp add*: *D-restriction-process*$_{ptick}I$)
next
fix $v'\ r$ **assume** ‹$v = v' @ [\checkmark(r)]$› ‹$tF\ v'$›
with ‹$v \in \mathcal{D}\ Q'$› ‹*length* $t \leq n$› ‹$t = u @ v$›
have ‹$v' \in \mathcal{D}\ Q'$› ‹*length* $v' < n$› **by** (*auto intro*: *is-processT9*)
with ‹$Q \downarrow n = Q' \downarrow n$› **have** ‹$v' \in \mathcal{D}\ Q$›
by (*metis D-restriction-process*$_{ptick}I$
length-less-in-D-restriction-process$_{ptick}$)
with ‹$v = v' @ [\checkmark(r)]$› ‹$tF\ v'$› **have** ‹$v \in \mathcal{D}\ Q$› **by** (*simp add*: *is-processT7*)
with ‹$tF\ u$› ‹$u \in \mathcal{T}\ P$› **have** ‹$t \in \mathcal{D}\ (P \triangle Q)$›
by (*auto simp add*: ‹$t = u @ v$› *D-Interrupt*)
thus ‹$t \in \mathcal{D}$ *?lhs*› **by** (*simp add*: *D-restriction-process*$_{ptick}I$)
qed
next
assume ‹*length* $v \neq n$›
with ‹*length* $t \leq n$› ‹$t = u @ v$› **have** ‹*length* $v < n$› **by** *simp*
with ‹$Q \downarrow n = Q' \downarrow n$› ‹$v \in \mathcal{D}\ Q'$› **have** ‹$v \in \mathcal{D}\ Q$›

```
          by (metis D-restriction-process_ptick I
              length-less-in-D-restriction-process_ptick)
        with ‹t = u @ v› ‹tF u› ‹u ∈ 𝒯 P› have ‹t ∈ 𝒟 (P △ Q)›
          by (auto simp add: D-Interrupt)
        thus ‹t ∈ 𝒟 ?lhs› by (simp add: D-restriction-process_ptick I)
      qed
    qed
  next
    show ‹t = u @ v ⟹ u ∈ 𝒯 (P' △ Q') ⟹ length u = n ⟹
        tF u ⟹ ftF v ⟹ t ∈ 𝒟 ?lhs› for u v by (fact *)
  qed

  show ‹(t, X) ∈ ℱ ?rhs ⟹ (t, X) ∈ ℱ ?lhs› for t X
  proof (elim F-restriction-process_ptick E)
    assume ‹(t, X) ∈ ℱ (P' △ Q')› ‹length t ≤ n›
    from this(1) consider ‹t ∈ 𝒟 (P' △ Q')›
      | u r where ‹t = u @ [✓(r)]› ‹u @ [✓(r)] ∈ 𝒯 P'›
      | r where ‹✓(r) ∉ X› ‹t @ [✓(r)] ∈ 𝒯 P'›
      | ‹(t, X) ∈ ℱ P'› ‹tF t› ‹([], X) ∈ ℱ Q'›
      | u v where ‹t = u @ v› ‹u ∈ 𝒯 P'› ‹tF u› ‹(v, X) ∈ ℱ Q'›
‹v ≠ []›
      | r where ‹✓(r) ∉ X› ‹t ∈ 𝒯 P'› ‹tF t› ‹[✓(r)] ∈ 𝒯 Q'›
      unfolding Interrupt-projs by blast
    thus ‹(t, X) ∈ ℱ ?lhs›
    proof cases
      assume ‹t ∈ 𝒟 (P' △ Q')›
      hence ‹t ∈ 𝒟 ?rhs› by (simp add: D-restriction-process_ptick I)
      with div D-F show ‹t ∈ 𝒟 (P' △ Q') ⟹ (t, X) ∈ ℱ ?lhs›
by blast
    next
      fix u r assume ‹t = u @ [✓(r)]› ‹u @ [✓(r)] ∈ 𝒯 P'›
      with ‹P ↓ n = P' ↓ n› ‹length t ≤ n› have ‹u @ [✓(r)] ∈ 𝒯
P›
      by (metis T-restriction-process_ptick I length-le-in-T-restriction-process_ptick)
      hence ‹(t, X) ∈ ℱ (P △ Q)› by (auto simp add: ‹t = u @
[✓(r)]› F-Interrupt)
    thus ‹(t, X) ∈ ℱ ?lhs› by (simp add: F-restriction-process_ptick I)
    next
      fix r assume ‹✓(r) ∉ X› ‹t @ [✓(r)] ∈ 𝒯 P'›
      show ‹(t, X) ∈ ℱ ?lhs›
      proof (cases ‹length t = n›)
        assume ‹length t = n›
        with ‹P ↓ n = P' ↓ n› ‹length t ≤ n› ‹t @ [✓(r)] ∈ 𝒯 P'›
have ‹t ∈ 𝒯 P›
          by (metis T-restriction-process_ptick I is-processT3-TR-append
              length-le-in-T-restriction-process_ptick)
        moreover from ‹t @ [✓(r)] ∈ 𝒯 P'› append-T-imp-tickFree
have ‹tF t› by blast
        ultimately have ‹t ∈ 𝒯 (P △ Q)› by (simp add: T-Interrupt)
```

with ‹*length t* = *n*› ‹*tF t*› **show** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs*›
by (*simp add*: *F-restriction-process*$_{ptick}$*I*)
next
assume ‹*length t* ≠ *n*›
with ‹*length t* ≤ *n*› **have** ‹*length* (*t* @ [✓(*r*)]) ≤ *n*› **by** *simp*
with ‹*P* ↓ *n* = *P'* ↓ *n*› ‹*t* @ [✓(*r*)] ∈ $\mathcal{T}$ *P'*› **have** ‹*t* @ [✓(*r*)]
∈ $\mathcal{T}$ *P*›
by (*metis T-restriction-process*$_{ptick}$*I length-le-in-T-restriction-process*$_{ptick}$)
with ‹✓(*r*) ∉ *X*› **have** ‹(*t*, *X*) ∈ $\mathcal{F}$ (*P* △ *Q*)›
by (*simp add*: *F-Interrupt*) (*metis Diff-insert-absorb*)
thus ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs*› **by** (*simp add*: *F-restriction-process*$_{ptick}$*I*)
qed
next
assume ‹(*t*, *X*) ∈ $\mathcal{F}$ *P'*› ‹*tF t*› ‹([], *X*) ∈ $\mathcal{F}$ *Q'*›
show ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs*›
proof (*cases* ‹*length t* = *n*›)
assume ‹*length t* = *n*›
from ‹(*t*, *X*) ∈ $\mathcal{F}$ *P'*› ‹*P* ↓ *n* = *P'* ↓ *n*› ‹*length t* ≤ *n*› **have**
‹*t* ∈ $\mathcal{T}$ *P*›
by (*simp add*: *F-T T-restriction-process*$_{ptick}$
length-le-in-T-restriction-process$_{ptick}$)
hence ‹*t* ∈ $\mathcal{T}$ (*P* △ *Q*)› **by** (*simp add*: *T-Interrupt*)
with ‹*length t* = *n*› ‹*tF t*› **show** ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs*›
by (*simp add*: *F-restriction-process*$_{ptick}$*I*)
next
assume ‹*length t* ≠ *n*›
with ‹*length t* ≤ *n*› **have** ‹*length t* < *n*› **by** *simp*
with ‹(*t*, *X*) ∈ $\mathcal{F}$ *P'*› ‹*P* ↓ *n* = *P'* ↓ *n*› ‹*length t* ≠ *n*› **have**
‹(*t*, *X*) ∈ $\mathcal{F}$ *P*›
by (*metis F-restriction-process*$_{ptick}$*I length-less-in-F-restriction-process*$_{ptick}$)
moreover from ‹([], *X*) ∈ $\mathcal{F}$ *Q'*› **have** ‹([], *X*) ∈ $\mathcal{F}$ *Q*›
by (*metis F-restriction-process*$_{ptick}$*I* ‹*0* < *n*› ‹*Q* ↓ *n* = *Q'*
↓ *n*›
length-less-in-F-restriction-process$_{ptick}$ *list.size*(*3*))
ultimately have ‹(*t*, *X*) ∈ $\mathcal{F}$ (*P* △ *Q*)›
using ‹*tF t*› **by** (*simp add*: *F-Interrupt*)
thus ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs*› **by** (*simp add*: *F-restriction-process*$_{ptick}$*I*)
qed
next
fix *u v* **assume** ‹*t* = *u* @ *v*› ‹*u* ∈ $\mathcal{T}$ *P'*› ‹*tF u*› ‹(*v*, *X*) ∈ $\mathcal{F}$
Q'› ‹*v* ≠ []›
from ‹*t* = *u* @ *v*› ‹*length t* ≤ *n*› **have** ‹*length u* ≤ *n*› **by** *simp*
with ‹*P* ↓ *n* = *P'* ↓ *n*› ‹*u* ∈ $\mathcal{T}$ *P'*› **have** ‹*u* ∈ $\mathcal{T}$ *P*›
by (*simp add*: *T-restriction-process*$_{ptick}$ *length-le-in-T-restriction-process*$_{ptick}$)
show ‹(*t*, *X*) ∈ $\mathcal{F}$ *?lhs*›
proof (*cases* ‹*length v* = *n*›)
assume ‹*length v* = *n*›
with ‹*t* = *u* @ *v*› ‹*length t* ≤ *n*› **have** ‹*u* = []› **by** *simp*
from *F-imp-front-tickFree* ‹(*v*, *X*) ∈ $\mathcal{F}$ *Q'*› **have** ‹*ftF v*› **by**

blast
thus ‹$(t, X) \in \mathcal{F}$ *?lhs*›
proof (*elim front-tickFreeE*)
assume ‹*tF v*›
from ‹*length* $v = n$› ‹$Q \downarrow n = Q' \downarrow n$› ‹$(v, X) \in \mathcal{F}\ Q'$›
have ‹$v \in \mathcal{T}\ Q$›
by (*metis F-T F-restriction-process*$_{ptick}$*I dual-order.refl*
length-le-in-T-restriction-process$_{ptick}$)
with ‹*tF u*› ‹$u \in \mathcal{T}\ P$› **have** ‹$t \in \mathcal{T}\ (P \triangle Q)$›
by (*auto simp add*: ‹$t = u \mathbin{@} v$› *T-Interrupt*)
with ‹*length* $v = n$› ‹$t = u \mathbin{@} v$› ‹*tF v*› ‹$u = []$› **show** ‹$(t, X) \in \mathcal{F}$ *?lhs*›
by (*simp add*: *F-restriction-process*$_{ptick}$*I*)
next
fix v' r **assume** ‹$v = v' \mathbin{@} [\checkmark(r)]$› ‹*tF v'*›
with ‹$(v, X) \in \mathcal{F}\ Q'$› ‹*length* $t \leq n$› ‹$t = u \mathbin{@} v$›
‹$Q \downarrow n = Q' \downarrow n$› ‹$u = []$› **have** ‹$v' \mathbin{@} [\checkmark(r)] \in \mathcal{T}\ Q$›
by (*metis F-T T-restriction-process*$_{ptick}$*I append-self-conv2*
length-le-in-T-restriction-process$_{ptick}$)
with ‹$t = u \mathbin{@} v$› ‹*tF u*› ‹$u \in \mathcal{T}\ P$› ‹$v = v' \mathbin{@} [\checkmark(r)]$› **have**
‹$t \in \mathcal{T}\ (P \triangle Q)$›
by (*auto simp add*: *T-Interrupt*)
thus ‹$(t, X) \in \mathcal{F}$ *?lhs*›
by (*simp add*: ‹$t = u \mathbin{@} v$› ‹$u = []$› ‹$v = v' \mathbin{@} [\checkmark(r)]$›
restriction-process$_{ptick}$*-projs*(*3*) *tick-T-F*)
qed
next
assume ‹*length* $v \neq n$›
with ‹*length* $t \leq n$› ‹$t = u \mathbin{@} v$› **have** ‹*length* $v < n$› **by** *simp*
with ‹$Q \downarrow n = Q' \downarrow n$› ‹$(v, X) \in \mathcal{F}\ Q'$› **have** ‹$(v, X) \in \mathcal{F}\ Q$›
by (*metis F-restriction-process*$_{ptick}$*I*
length-less-in-F-restriction-process$_{ptick}$)
with ‹$t = u \mathbin{@} v$› ‹*tF u*› ‹$u \in \mathcal{T}\ P$› ‹$v \neq []$› **have** ‹$(t, X) \in \mathcal{F}\ (P \triangle Q)$›
by (*simp add*: *F-Interrupt*) *blast*
thus ‹$(t, X) \in \mathcal{F}$ *?lhs*› **by** (*simp add*: *F-restriction-process*$_{ptick}$*I*)
qed
next
fix r **assume** ‹$\checkmark(r) \notin X$› ‹$t \in \mathcal{T}\ P'$› ‹*tF t*› ‹$[\checkmark(r)] \in \mathcal{T}\ Q'$›
have ‹$t \in \mathcal{T}\ P$›
by (*metis T-restriction-process*$_{ptick}$*I* ‹$P \downarrow n = P' \downarrow n$› ‹*length* $t \leq n$›
‹$t \in \mathcal{T}\ P'$› *length-le-in-T-restriction-process*$_{ptick}$)
moreover have ‹$[\checkmark(r)] \in \mathcal{T}\ Q$›
by (*metis* (*no-types*, *lifting*) *Suc-leI T-restriction-process*$_{ptick}$
Un-iff ‹*0* $< n$› ‹$Q \downarrow n = Q' \downarrow n$› ‹$[\checkmark(r)] \in \mathcal{T}\ Q'$› *length-Cons*
length-le-in-T-restriction-process$_{ptick}$ *list.size*(*3*))
ultimately have ‹$(t, X) \in \mathcal{F}\ (P \triangle Q)$›
using ‹$\checkmark(r) \notin X$› ‹*tF t*› **by** (*simp add*: *F-Interrupt*) *blast*

thus ‹$(t, X) \in \mathcal{F}$ *?lhs*› **by** (*simp add*: *F-restriction-process*$_{ptick}I$)
qed
next
show ‹$t = u$ @ $v \Longrightarrow u \in \mathcal{T}$ $(P' \mathbin{\square} Q') \Longrightarrow$ *length* $u = n \Longrightarrow$
tF $u \Longrightarrow$ *ftF* $v \Longrightarrow (t, X) \in \mathcal{F}$ *?lhs*› **for** u v
by (*simp add*: $*$ *is-processT8*)
qed
qed
qed

9 Non too Destructiveness of After

9.1 Equality

lemma *initials-restriction-process*$_{ptick}$: ‹$(P \downarrow n)^0 =$ (*if* $n = 0$ *then* *UNIV else* P^0)›
by (*cases n*, *solves simp*)
(*auto simp add*: *initials-def T-restriction-process*$_{ptick}$,
metis append.right-neutral append-eq-conv-conj drop-Nil drop-Suc-Cons)

lemma (**in** *After*) *restriction-process*$_{ptick}$*-After*:
‹P *after* $e \downarrow n =$ (*if ev* $e \in P^0$ *then* $(P \downarrow$ *Suc* $n)$ *after* e *else* Ψ P e $\downarrow n$)›
proof (*split if-split*, *intro conjI impI*)
show ‹*ev* $e \notin P^0 \Longrightarrow P$ *after* $e \downarrow n = \Psi$ P $e \downarrow n$› **by** (*simp add*: *not-initial-After*)
next
assume ‹*ev* $e \in P^0$›
show ‹P *after* $e \downarrow n = (P \downarrow$ *Suc* $n)$ *after* e› (**is** ‹*?lhs* = *?rhs*›)
proof (*subst Process-eq-spec*, *safe*)
show ‹$t \in \mathcal{D}$ *?lhs* $\Longrightarrow t \in \mathcal{D}$ *?rhs*› **for** t
by (*elim D-restriction-process*$_{ptick}E$)
(*simp-all add*: ‹*ev* $e \in P^0$› *After-projs*
initials-restriction-process$_{ptick}$ *D-restriction-process*$_{ptick}$,
meson Cons-eq-appendI event$_{ptick}$*.disc*(*1*) *length-Cons tick-Free-Cons-iff*)
next
show ‹$t \in \mathcal{D}$ *?rhs* $\Longrightarrow t \in \mathcal{D}$ *?lhs*› **for** t
by (*auto simp add*: *D-After initials-restriction-process*$_{ptick}$ ‹*ev* $e \in P^0$›
D-restriction-process$_{ptick}$ *T-After Cons-eq-append-conv*)
next
show ‹$(t, X) \in \mathcal{F}$ *?lhs* $\Longrightarrow (t, X) \in \mathcal{F}$ *?rhs*› **for** t X
by (*elim F-restriction-process*$_{ptick}E$)
(*simp-all add*: ‹*ev* $e \in P^0$› *After-projs*
initials-restriction-process$_{ptick}$ *F-restriction-process*$_{ptick}$,

meson Cons-eq-appendI event$_{ptick}$.disc(1) length-Cons tick-Free-Cons-iff)
next
show ‹(*t*, *X*) ∈ $\mathcal{F}$ *?rhs* ⟹ (*t*, *X*) ∈ $\mathcal{F}$ *?lhs*› **for** *t X*
by (*auto simp add*: *F-After initials-restriction-process$_{ptick}$* ‹*ev e* ∈ P^0›
F-restriction-process$_{ptick}$ T-After Cons-eq-append-conv)
qed
qed

lemma (**in** *AfterExt*) *restriction-process$_{ptick}$-After$_{tick}$*:
‹*P after*$_{✓}$ *e* ↓ *n* =
(*case e of* ✓(*r*) ⇒ Ω *P r* ↓ *n* | *ev x* ⇒ *if e* ∈ P^0 *then* (*P* ↓ *Suc n*) *after*$_{✓}$ *e else* Ψ *P x* ↓ *n*)›
by (*simp add*: *After$_{tick}$-def restriction-process$_{ptick}$-After split*: *event$_{ptick}$.split*)

lemma (**in** *AfterExt*) *restriction-process$_{ptick}$-After$_{trace}$*:
‹*t* ∈ $\mathcal{T}$ *P* ⟹ *tF t* ⟹ *P after*$_{\mathcal{T}}$ *t* ↓ *n* = (*P* ↓ (*n* + *length t*)) *after*$_{\mathcal{T}}$ *t*›
proof (*induct t arbitrary*: *n rule*: *rev-induct*)
show ‹*P after*$_{\mathcal{T}}$ [] ↓ *n* = (*P* ↓ (*n* + *length* [])) *after*$_{\mathcal{T}}$ []› **for** *n* **by** *simp*
next
fix *e t n*
assume *hyp* : ‹*t* ∈ $\mathcal{T}$ *P* ⟹ *tF t* ⟹ *P after*$_{\mathcal{T}}$ *t* ↓ *n* = (*P* ↓ (*n* + *length t*)) *after*$_{\mathcal{T}}$ *t*› **for** *n*
assume *prems* : ‹*t* @ [*e*] ∈ $\mathcal{T}$ *P*› ‹*tickFree* (*t* @ [*e*])›
from *prems*(*2*) **obtain** *a* **where** ‹*e* = *ev a*› **by** (*cases e*) *simp-all*
with *initials-After$_{trace}$*[*OF prems*(*1*)] **have** ‹*ev a* ∈ (*P after*$_{\mathcal{T}}$ *t*)^0› **by** *simp*
from *prems is-processT3-TR-append* **have** ‹*t* ∈ $\mathcal{T}$ *P*› ‹*tF t*› **by** *auto*
from *hyp*[*OF this*] **have** ‹*P after*$_{\mathcal{T}}$ *t* ↓ *Suc n* = (*P* ↓ (*Suc n* + *length t*)) *after*$_{\mathcal{T}}$ *t*› **.**
thus ‹*P after*$_{\mathcal{T}}$ (*t* @ [*e*]) ↓ *n* = (*P* ↓ (*n* + *length* (*t* @ [*e*]))) *after*$_{\mathcal{T}}$ (*t* @ [*e*])›
by (*simp add*: *After$_{trace}$-snoc restriction-process$_{ptick}$-After$_{tick}$* ‹*e* = *ev a*› ‹*ev a* ∈ (*P after*$_{\mathcal{T}}$ *t*)^0›)
qed

9.2 Non too Destructiveness

lemma (**in** *After*) *non-too-destructive-on-After* :
‹*non-too-destructive-on* (λ*P*. *P after e*) {*P*. *ev e* ∈ P^0}›
by (*auto intro*!: *non-too-destructive-onI simp add*: *restriction-process$_{ptick}$-After*)

lemma (**in** *AfterExt*) *non-too-destructive-on-After$_{tick}$* :
‹*non-too-destructive-on* (λ*P*. *P after*$_{✓}$ *e*) {*P*. *e* ∈ P^0}›
if ‹⋀*r*. *e* = ✓(*r*) ⟹ *non-too-destructive-on* Ω {*P*. ✓(*r*) ∈ P^0}›

proof (*intro non-too-destructive-onI*, *clarify*)
fix P Q n **assume** $*$: ‹$P \downarrow Suc\ n = Q \downarrow Suc\ n$› ‹$e \in P^0$› ‹$e \in Q^0$›
show ‹$P\ after_{\checkmark}\ e \downarrow n = Q\ after_{\checkmark}\ e \downarrow n$›
proof (*cases e*)
from $*$ **show** ‹$e = ev\ a \Longrightarrow P\ after_{\checkmark}\ e \downarrow n = Q\ after_{\checkmark}\ e \downarrow n$› **for** *a*
by (*simp add*: *restriction-process*$_{ptick}$-*After*$_{tick}$)
next
fix r **assume** ‹$e = \checkmark(r)$›
with $*$(*2*, *3*) **have** ‹$P \in \{P.\ \checkmark(r) \in P^0\}$› ‹$Q \in \{P.\ \checkmark(r) \in P^0\}$›
by *auto*
from *non-too-destructive-onD*[*OF that*[*simplified* ‹$e = \checkmark(r)$›, *OF refl*] *this* $*$(*1*)]
have ‹$\Omega\ P \downarrow n = \Omega\ Q \downarrow n$› **.**
with ‹$e = \checkmark(r)$› **show** ‹$P\ after_{\checkmark}\ e \downarrow n = Q\ after_{\checkmark}\ e \downarrow n$›
by (*simp add*: *restriction-process*$_{ptick}$-*After*$_{tick}$) (*metis restriction-fun-def*)
qed
qed

lemma (**in** *After*) *non-too-destructive-After* :
‹*non-too-destructive* ($\lambda P.\ P\ after\ e$)› **if** $*$: ‹*non-too-destructive-on* Ψ $\{P.\ ev\ e \notin P^0\}$›
proof (*rule non-too-destructiveI*)
fix P Q :: ‹('*a*, '*r*) *process*$_{ptick}$› **and** n
assume ‹$P \downarrow Suc\ n = Q \downarrow Suc\ n$›
hence ‹$ev\ e \in P^0 \wedge ev\ e \in Q^0 \vee ev\ e \notin P^0 \wedge ev\ e \notin Q^0$›
by (*metis initials-restriction-process*$_{ptick}$ *nat.distinct*(*1*))
thus ‹$P\ after\ e \downarrow n = Q\ after\ e \downarrow n$›
proof (*elim disjE conjE*)
show ‹$ev\ e \in P^0 \Longrightarrow ev\ e \in Q^0 \Longrightarrow P\ after\ e \downarrow n = Q\ after\ e \downarrow n$›
by (*simp add*: *restriction-process*$_{ptick}$-*After* ‹$P \downarrow Suc\ n = Q \downarrow Suc\ n$›)
next
assume ‹$ev\ e \notin P^0$› ‹$ev\ e \notin Q^0$›
hence ‹$P\ after\ e = \Psi\ P\ e$› ‹$Q\ after\ e = \Psi\ Q\ e$›
by (*simp-all add*: *not-initial-After*)
from ‹$P \downarrow Suc\ n = Q \downarrow Suc\ n$› **have** ‹$\Psi\ P \downarrow n = \Psi\ Q \downarrow n$›
by (*intro* $*$[*THEN non-too-destructive-onD*, *of P Q*])
(*simp-all add*: ‹$ev\ e \notin P^0$› ‹$ev\ e \notin Q^0$›)
with ‹$P\ after\ e = \Psi\ P\ e$› ‹$Q\ after\ e = \Psi\ Q\ e$›
show ‹$P\ after\ e \downarrow n = Q\ after\ e \downarrow n$›
by (*metis restriction-fun-def*)
qed
qed

lemma (**in** *AfterExt*) *non-too-destructive-After*$_{tick}$:
‹*non-too-destructive* (λ*P*. *P after*$_{✓}$ *e*)›
if ∗ : ‹⋀*a*. *e* = *ev a* ⟹ *non-too-destructive-on* Ψ {*P*. *ev a* ∉ *P*^{0}}›
‹⋀*r*. *e* = ✓(*r*) ⟹ *non-too-destructive* (λ*P*. Ω *P r*)›
proof (*rule non-too-destructiveI*)
show ‹*P after*$_{✓}$ *e* ↓ *n* = *Q after*$_{✓}$ *e* ↓ *n*› **if** ‹*P* ↓ *Suc n* = *Q* ↓ *Suc n*› **for** *P Q n*
proof (*cases e*)
show ‹*P after*$_{✓}$ *e* ↓ *n* = *Q after*$_{✓}$ *e* ↓ *n*› **if** ‹*e* = *ev a*› **for** *a*
by (*simp add*: *After*$_{tick}$*-def* ‹*e* = *ev a*›)
(*fact non-too-destructive-After*[*OF* ∗(*1*)[*simplified* ‹*e* = *ev a*›, *OF refl*],
THEN non-too-destructiveD, *OF* ‹*P* ↓ *Suc n* = *Q* ↓ *Suc n*›])
next
fix *r* **assume** ‹*e* = ✓(*r*)›
from ∗(*2*)[*unfolded* ‹*e* = ✓(*r*)›, *OF refl*,
THEN non-too-destructiveD, *OF* ‹*P* ↓ *Suc n* = *Q* ↓ *Suc n*›]
have ‹Ω *P r* ↓ *n* = Ω *Q r* ↓ *n*› **.**
thus ‹*P after*$_{✓}$ *e* ↓ *n* = *Q after*$_{✓}$ *e* ↓ *n*›
by (*simp add*: *After*$_{tick}$*-def* ‹*e* = ✓(*r*)›)
qed
qed

lemma (**in** *AfterExt*) *restriction-shift-After*$_{trace}$:
‹*restriction-shift* (λ*P*. *P after*$_{\mathcal{T}}$ *t*) (− *int* (*length t*))›
if ‹*non-too-destructive* Ψ› ‹*non-too-destructive* Ω›
— We could imagine more precise assumptions, but is it useful?
proof (*induct t*)
case *Nil* **show** *?case* **by** (*simp add*: *restriction-shiftI*)
next
case (*Cons e t*)
have ‹*non-too-destructive* (λ*P*. *P after*$_{✓}$ *e*)›
by (*rule non-too-destructive-After*$_{tick}$)
(*simp add*: ‹*non-too-destructive* Ψ›,
metis non-too-destructive-fun-iff ‹*non-too-destructive* Ω›)
hence ∗ : ‹*restriction-shift* (λ*P*. *P after*$_{✓}$ *e*) (− *1*)›
unfolding *non-too-destructive-def*
non-too-destructive-on-def restriction-shift-def **.**
from *restriction-shift-comp-restriction-shift*[*OF Cons.hyps* ∗]
show *?case* **by** *simp*
qed

10 Destructiveness of Hiding

theory *Hiding-Destructive*
imports *HOL−CSPM.CSPM-Laws Prefixes-Constructive*
begin

10.1 Refinement

lemma *Hiding-restriction-process$_{ptick}$-FD* : ‹$(P \downarrow n) \setminus S \sqsubseteq_{FD} P \setminus S \downarrow n$›
proof (*unfold refine-defs*, *safe*)
show $*$: ‹$t \in \mathcal{D}\ (P \setminus S \downarrow n) \Longrightarrow t \in \mathcal{D}\ ((P \downarrow n) \setminus S)$› **for** t
proof (*elim D-restriction-process$_{ptick}$E*)
show ‹$t \in \mathcal{D}\ (P \setminus S) \Longrightarrow t \in \mathcal{D}\ ((P \downarrow n) \setminus S)$›
by (*simp add*: *D-Hiding restriction-process$_{ptick}$-projs*) *blast*
next
fix $u\ v$
assume ‹$t = u\ @\ v$› ‹$u \in \mathcal{T}\ (P \setminus S)$› ‹*length* $u = n$› ‹*tF* u› ‹*ftF* v›
from ‹$u \in \mathcal{T}\ (P \setminus S)$›[*simplified T-Hiding*, *simplified*]
consider ‹$u \in \mathcal{D}\ (P \setminus S)$› | u' **where** ‹$u =$ *trace-hide* $u'\ (ev\ `\ S)$› ‹$(u', ev\ `\ S) \in \mathcal{F}\ P$›
by (*simp add*: *F-Hiding D-Hiding*) *blast*
thus ‹$t \in \mathcal{D}\ ((P \downarrow n) \setminus S)$›
proof *cases*
assume ‹$u \in \mathcal{D}\ (P \setminus S)$›
hence ‹$t \in \mathcal{D}\ (P \setminus S)$› **by** (*simp add*: ‹*ftF* v› ‹$t = u\ @\ v$› ‹*tF* u› *is-processT7*)
with *restriction-process$_{ptick}$-approx-self le-approx1 mono-Hiding*
show ‹$t \in \mathcal{D}\ ((P \downarrow n) \setminus S)$› **by** *blast*
next
fix u' **assume** ‹$u =$ *trace-hide* $u'\ (ev\ `\ S)$› ‹$(u', ev\ `\ S) \in \mathcal{F}\ P$›
with ‹*length* $u = n$› ‹*tF* u› *Hiding-tickFree length-filter-le F-T*
have ‹$n \leq$ *length* u'› ‹*tickFree* u'› ‹$u' \in \mathcal{T}\ P$› **by** *blast+*
with ‹$u =$ *trace-hide* $u'\ (ev\ `\ S)$›
have ‹$u' = (take\ n\ u')\ @\ (drop\ n\ u') \wedge take\ n\ u' \in \mathcal{T}\ P \wedge$
length $(take\ n\ u') = n \wedge tF\ (take\ n\ u') \wedge ftF\ (drop\ n\ u')$›
by (*simp add*: *min-def*) (*metis append-take-drop-id is-processT3-TR-append tickFree-append-iff tickFree-imp-front-tickFree*)
with *D-restriction-process$_{ptick}$* **have** ‹$u' \in \mathcal{D}\ (P \downarrow n)$› **by** *blast*
with *Hiding-tickFree* ‹*ftF* v› ‹$t = u\ @\ v$› ‹$u =$ *trace-hide* $u'\ (ev\ `\ S)$› ‹*tF* u›
show ‹$t \in \mathcal{D}\ ((P \downarrow n) \setminus S)$› **by** (*simp add*: *D-Hiding*) *blast*
qed
qed

fix $s\ X$
assume ‹$(s, X) \in \mathcal{F}\ ((P \setminus S) \downarrow n)$›
then consider ‹$s \in \mathcal{D}\ ((P \setminus S) \downarrow n)$› | ‹$(s, X) \in \mathcal{F}\ (P \setminus S)$›
unfolding *restriction-process$_{ptick}$-projs* **by** *blast*

thus ‹$(s,\ X) \in \mathcal{F}\ ((P \downarrow n) \setminus S)$›
proof *cases*
from $*$ *D-F* **show** ‹$s \in \mathcal{D}\ ((P \setminus S) \downarrow n) \Longrightarrow (s,\ X) \in \mathcal{F}\ ((P \downarrow n) \setminus S)$› **by** *blast*
next
show ‹$(s,\ X) \in \mathcal{F}\ (P \setminus S) \Longrightarrow (s,\ X) \in \mathcal{F}\ ((P \downarrow n) \setminus S)$›
using *restriction-process$_{ptick}$-approx-self D-F mono-Hiding proc-ord2a*
by *blast*
qed
qed

10.2 Destructiveness

lemma *Hiding-destructive* :
‹$\exists\, P\ Q :: ('a,\ 'r)\ process_{ptick}.\ P \downarrow n = Q \downarrow n \wedge (P \setminus S) \downarrow Suc\ 0 \neq (Q \setminus S) \downarrow Suc\ 0$› **if** ‹$S \neq \{\}$›
proof −
from ‹$S \neq \{\}$› **obtain** *e* **where** ‹$e \in S$› **by** *blast*
define $P :: ‹('a,\ 'r)\ process_{ptick}›$ **where** ‹$P \equiv iterate\ (Suc\ n)\cdot(\Lambda\ X.\ write0\ e\ X)\cdot(SKIP\ undefined)$›
define $Q :: ‹('a,\ 'r)\ process_{ptick}›$ **where** ‹$Q \equiv iterate\ (Suc\ n)\cdot(\Lambda\ X.\ write0\ e\ X)\cdot STOP$›
have ‹$P \downarrow n = Q \downarrow n$›
unfolding *P-def Q-def* **by** (*induct n*) (*simp-all add*: *restriction-process$_{ptick}$-write0*)

have ‹$P \setminus S = SKIP\ undefined$›
unfolding *P-def* **by** (*induct n*) (*simp-all add*: *Hiding-write0-non-disjoint* ‹$e \in S$›)
hence ‹$(P \setminus S) \downarrow Suc\ 0 = SKIP\ undefined$› **by** *simp*
have ‹$Q \setminus S = STOP$›
unfolding *Q-def* **by** (*induct n*) (*simp-all add*: *Hiding-write0-non-disjoint* ‹$e \in S$›)
hence ‹$(Q \setminus S) \downarrow Suc\ 0 = STOP$› **by** *simp*

have ‹$P \downarrow n = Q \downarrow n \wedge (P \setminus S) \downarrow Suc\ 0 \neq (Q \setminus S) \downarrow Suc\ 0$›
by (*simp add*: ‹$P \downarrow n = Q \downarrow n$› ‹$(P \setminus S) \downarrow Suc\ 0 = SKIP\ undefined$› ‹$(Q \setminus S) \downarrow Suc\ 0 = STOP$› *SKIP-Neq-STOP*)
thus *?thesis* **by** *blast*
qed

11 Admissibility

named-theorems *restriction-adm-process$_{ptick}$-simpset*

11.1 Belonging

lemma *restriction-adm-in-D* [*restriction-adm-process$_{ptick}$-simpset*] :
‹$adm_{\downarrow}$ ($\lambda x.\ t \in \mathcal{D}\ (f\ x)$)›
and *restriction-adm-notin-D* [*restriction-adm-process$_{ptick}$-simpset*] :
‹$adm_{\downarrow}$ ($\lambda x.\ t \notin \mathcal{D}\ (f\ x)$)›
and *restriction-adm-in-F* [*restriction-adm-process$_{ptick}$-simpset*] :
‹$adm_{\downarrow}$ ($\lambda x.\ (t,\ X) \in \mathcal{F}\ (f\ x)$)›
and *restriction-adm-notin-F* [*restriction-adm-process$_{ptick}$-simpset*] :
‹$adm_{\downarrow}$ ($\lambda x.\ (t,\ X) \notin \mathcal{F}\ (f\ x)$)› **if** ‹$cont_{\downarrow}\ f$›
for f :: ‹$'b$ :: *restriction* $\Rightarrow$ ($'a$, $'r$) *process$_{ptick}$*›
proof (*all* ‹*rule restriction-adm-subst*[*OF* ‹$cont_{\downarrow}\ f$›]›)
have $*$: ‹σ $-\downarrow\rightarrow$ $\Sigma \Longrightarrow \exists\, n0.\ \forall\, n \geq n0.\ \sigma\ n \downarrow Suc\ (length\ t) = \Sigma \downarrow Suc\ (length\ t)$›
for σ **and** Σ :: ‹($'a$, $'r$) *process$_{ptick}$*›
by (*metis restriction-tendstoD*)

show ‹$adm_{\downarrow}$ ($\lambda x.\ t \in \mathcal{D}\ x$)› ‹$adm_{\downarrow}$ ($\lambda x.\ t \notin \mathcal{D}\ x$)›
by (*rule restriction-admI*,
metis (*no-types*) $*$ *D-restriction-process$_{ptick}$-Suc-length-iff-D dual-order.refl*)+

show ‹$adm_{\downarrow}$ ($\lambda x.\ (t,\ X) \in \mathcal{F}\ x$)› ‹$adm_{\downarrow}$ ($\lambda x.\ (t,\ X) \notin \mathcal{F}\ x$)›
by (*rule restriction-admI*,
metis (*no-types*) $*$ *F-restriction-process$_{ptick}$-Suc-length-iff-F dual-order.refl*)+
qed

corollary *restriction-adm-in-T* [*restriction-adm-process$_{ptick}$-simpset*]
:
‹$cont_{\downarrow}\ f \Longrightarrow adm_{\downarrow}$ ($\lambda x.\ t \in \mathcal{T}\ (f\ x)$)›
and *restriction-adm-notin-T* [*restriction-adm-process$_{ptick}$-simpset*] :
‹$cont_{\downarrow}\ f \Longrightarrow adm_{\downarrow}$ ($\lambda x.\ t \notin \mathcal{T}\ (f\ x)$)›
by (*fact restriction-adm-in-F*[*of f t* ‹{}›, *simplified T-F-spec*])
(*fact restriction-adm-notin-F*[*of f t* ‹{}›, *simplified T-F-spec*])

corollary *restriction-adm-in-initials* [*restriction-adm-process$_{ptick}$-simpset*]
:
‹$cont_{\downarrow}\ f \Longrightarrow adm_{\downarrow}$ ($\lambda x.\ e \in (f\ x)^0$)›
and *restriction-adm-notin-initials* [*restriction-adm-process$_{ptick}$-simpset*]
:
‹$cont_{\downarrow}\ f \Longrightarrow adm_{\downarrow}$ ($\lambda x.\ e \notin (f\ x)^0$)›
by (*simp-all add*: *initials-def restriction-adm-in-T restriction-adm-notin-T*)

11.2 Refining

corollary *restriction-adm-leF* [*restriction-adm-process$_{ptick}$-simpset*] :
‹$cont_{\downarrow}\ f \Longrightarrow cont_{\downarrow}\ g \Longrightarrow adm_{\downarrow}$ ($\lambda x.\ f\ x \sqsubseteq_F g\ x$)›

by (*simp add*: *failure-refine-def subset-iff restriction-adm-process$_{ptick}$-simpset*)

corollary *restriction-adm-leD* [*restriction-adm-process$_{ptick}$-simpset*] :
‹*cont$_\downarrow$ f* ⟹ *cont$_\downarrow$ g* ⟹ *adm$_\downarrow$* (λ*x*. *f x* ⊑$_D$ *g x*)›
by (*simp add*: *divergence-refine-def subset-iff restriction-adm-process$_{ptick}$-simpset*)

corollary *restriction-adm-leT* [*restriction-adm-process$_{ptick}$-simpset*] :
‹*cont$_\downarrow$ f* ⟹ *cont$_\downarrow$ g* ⟹ *adm$_\downarrow$* (λ*x*. *f x* ⊑$_T$ *g x*)›
by (*simp add*: *trace-refine-def subset-iff restriction-adm-process$_{ptick}$-simpset*)

corollary *restriction-adm-leFD* [*restriction-adm-process$_{ptick}$-simpset*]
:
‹*cont$_\downarrow$ f* ⟹ *cont$_\downarrow$ g* ⟹ *adm$_\downarrow$* (λ*x*. *f x* ⊑$_{FD}$ *g x*)›
by (*simp add*: *failure-divergence-refine-def restriction-adm-process$_{ptick}$-simpset*)

corollary *restriction-adm-leDT* [*restriction-adm-process$_{ptick}$-simpset*]
:
‹*cont$_\downarrow$ f* ⟹ *cont$_\downarrow$ g* ⟹ *adm$_\downarrow$* (λ*x*. *f x* ⊑$_{DT}$ *g x*)›
by (*simp add*: *trace-divergence-refine-def restriction-adm-process$_{ptick}$-simpset*)

11.2.1 Transitions

lemma (**in** *After*) *restriction-cont-After* [*restriction-adm-simpset*] :
‹*cont$_\downarrow$* (λ*x*. *f x after a*)› **if** ‹*cont$_\downarrow$ f*› **and** ‹*cont$_\downarrow$* Ψ›
— We could imagine more precise assumptions, but is it useful?
proof (*rule restriction-cont-comp*[*OF* - ‹*cont$_\downarrow$ f*›])
show ‹*cont$_\downarrow$* (λ*P*. *P after a*)›
proof (*rule restriction-contI*)
show ‹(λ*n*. σ *n after a*) −↓→ Σ *after a*› **if** ‹σ −↓→ Σ› **for** σ Σ
proof (*rule restriction-tendstoI*)
fix *n*
from ‹σ −↓→ Σ› **obtain** *n0*
where ∗ : ‹∀ *k*≥*n0*. Σ ↓ *Suc n* = σ *k* ↓ *Suc n*›
by (*blast dest*: *restriction-tendstoD*)
consider ‹*ev a* ∈ Σ^0› ‹∀ *n*≥*Suc n0*. *ev a* ∈ (σ *n*)^0›
| ‹*ev a* ∉ Σ^0› ‹∀ *n*≥*Suc n0*. *ev a* ∉ (σ *n*)^0›
by (*metis* ∗ *Suc-leD initials-restriction-process$_{ptick}$ nat.distinct*(*1*))
thus ‹∃ *n0*. ∀ *k*≥*n0*. Σ *after a* ↓ *n* = σ *k after a* ↓ *n*›
proof *cases*
assume ‹*ev a* ∈ Σ^0› ‹∀ *n*≥*Suc n0*. *ev a* ∈ (σ *n*)^0›
hence ‹∀ *k*≥*Suc n0*. Σ *after a* ↓ *n* = σ *k after a* ↓ *n*›
by (*metis* ∗ *Suc-leD restriction-process$_{ptick}$-After*)
thus ‹∃ *n0*. ∀ *k*≥*n0*. Σ *after a* ↓ *n* = σ *k after a* ↓ *n*› **..**
next
assume ‹*ev a* ∉ Σ^0› ‹∀ *n*≥*Suc n0*. *ev a* ∉ (σ *n*)^0›
hence ‹Σ *after a* = Ψ Σ *a*› ‹∀ *k*≥*Suc n0*. σ *k after a* = Ψ (σ
k) *a*›
by (*simp-all add*: *not-initial-After*)
moreover from ‹*cont$_\downarrow$* Ψ›[*THEN restriction-contD*]

obtain *n1* **where** ‹$\forall k{\geq}n1.\ \Psi\ \Sigma\downarrow n = \Psi\ (\sigma\ k)\downarrow n$›
by (*blast intro*: ‹$\sigma\ -\downarrow\rightarrow\ \Sigma$› *dest*: *restriction-tendstoD*)
ultimately have ‹$\forall k{\geq}max\ n1\ (Suc\ n0).\ \Sigma\ after\ a\downarrow n = \sigma\ k\ after\ a\downarrow n$›
by *simp* (*metis restriction-fun-def*)
thus ‹$\exists n0.\ \forall k{\geq}n0.\ \Sigma\ after\ a\downarrow n = \sigma\ k\ after\ a\downarrow n$› **..**
qed
qed
qed
qed

lemma (**in** *AfterExt*) *restriction-cont-After*$_{tick}$ [*restriction-adm-simpset*]
:
‹$cont_\downarrow\ (\lambda x.\ f\ x\ after_{\checkmark}\ e)$› **if** ‹$cont_\downarrow\ f$› **and** ‹$cont_\downarrow\ \Psi$› **and** ‹$cont_\downarrow\ \Omega$›
— We could imagine more precise assumptions, but is it useful?
proof (*cases e*)
show ‹$e = ev\ a \Longrightarrow cont_\downarrow\ (\lambda x.\ f\ x\ after_{\checkmark}\ e)$› **for** *a*
by (*simp add*: *After*$_{tick}$*-def restriction-cont-After* ‹$cont_\downarrow\ f$› ‹$cont_\downarrow\ \Psi$›)
next
fix *r* **assume** ‹$e = \checkmark(r)$›
hence ‹$(\lambda x.\ f\ x\ after_{\checkmark}\ e) = (\lambda x.\ \Omega\ (f\ x)\ r)$› **by** (*simp add*: *After*$_{tick}$*-def*)
thus ‹$cont_\downarrow\ (\lambda x.\ f\ x\ after_{\checkmark}\ e)$›
by (*metis restriction-cont-comp restriction-cont-fun-imp that*(*1*,*3*))
qed

lemma (**in** *AfterExt*) *restriction-cont-After*$_{trace}$ [*restriction-adm-simpset*]
:
‹$cont_\downarrow\ (\lambda x.\ f\ x\ after_{\mathcal{T}}\ t)$› **if** ‹$cont_\downarrow\ f$› **and** ‹$cont_\downarrow\ \Psi$› **and** ‹$cont_\downarrow\ \Omega$›
— We could imagine more precise assumptions, but is it useful?
proof (*rule restriction-cont-comp*[*OF* - ‹$cont_\downarrow\ f$›])
show ‹$cont_\downarrow\ (\lambda P.\ P\ after_{\mathcal{T}}\ t)$›
proof (*induct t*)
show ‹$cont_\downarrow\ (\lambda P.\ P\ after_{\mathcal{T}}\ [])$› **by** *simp*
next
fix *e t* **assume** ‹$cont_\downarrow\ (\lambda P.\ P\ after_{\mathcal{T}}\ t)$›
show ‹$cont_\downarrow\ (\lambda P.\ P\ after_{\mathcal{T}}\ (e\ \#\ t))$›
by (*simp*, *rule restriction-cont-comp*[*OF* ‹$cont_\downarrow\ (\lambda P.\ P\ after_{\mathcal{T}}\ t)$›])
(*simp add*: *restriction-cont-After*$_{tick}$ ‹$cont_\downarrow\ \Psi$› ‹$cont_\downarrow\ \Omega$›)
qed
qed

lemma (**in** *OpSemTransitions*) *restriction-adm-weak-ev-trans* [*restriction-adm-process*$_{ptick}$*-simpset*]:
— Could be weakened to a continuity assumption on Ψ.
fixes *f g* :: ‹$'b$ :: *restriction* $\Rightarrow$ ($'a$, $'r$) *process*$_{ptick}$›
assumes τ*-trans-restriction-adm*:

‹$\bigwedge f\ g :: 'b \Rightarrow ('a, 'r)\ process_{ptick}.\ cont_{\downarrow}\ f \Longrightarrow cont_{\downarrow}\ g \Longrightarrow adm_{\downarrow}$
$(\lambda x.\ f\ x \rightsquigarrow_{\tau} g\ x)$›
and ‹$cont_{\downarrow}\ f$› **and** ‹$cont_{\downarrow}\ g$› **and** ‹$cont_{\downarrow}\ \Psi$› **and** ‹$cont_{\downarrow}\ \Omega$›
shows ‹$adm_{\downarrow}\ (\lambda x.\ f\ x \rightsquigarrow_{e} g\ x)$›
proof (*intro restriction-adm-conj*)
show ‹$adm_{\downarrow}\ (\lambda x.\ ev\ e \in (f\ x)^0)$›
by (*simp add*: ‹$cont_{\downarrow}\ f$› *restriction-adm-in-initials*)
next
show ‹$adm_{\downarrow}\ (\lambda x.\ f\ x\ after_{\checkmark}\ ev\ e \rightsquigarrow_{\tau} g\ x)$›
proof (*rule* τ*-trans-restriction-adm*[*OF* - ‹$cont_{\downarrow}\ g$›],
rule restriction-cont-comp[*OF* - ‹$cont_{\downarrow}\ f$›])
show ‹$cont_{\downarrow}\ (\lambda x.\ x\ after_{\checkmark}\ ev\ e)$›
by (*simp add*: ‹$cont_{\downarrow}\ \Psi$› ‹$cont_{\downarrow}\ \Omega$› *restriction-cont-After*$_{tick}$)
qed
qed

lemma (**in** *OpSemTransitions*) *restriction-adm-weak-tick-trans* [*restriction-adm-process*$_{ptick}$*-simpset*]:
fixes $f\ g ::$ ‹$'b ::$ *restriction* $\Rightarrow ('a, 'r)\ process_{ptick}$›
assumes τ*-trans-restriction-adm*:
‹$\bigwedge f\ g :: 'b \Rightarrow ('a, 'r)\ process_{ptick}.\ cont_{\downarrow}\ f \Longrightarrow cont_{\downarrow}\ g \Longrightarrow adm_{\downarrow}$
$(\lambda x.\ f\ x \rightsquigarrow_{\tau} g\ x)$›
and ‹$cont_{\downarrow}\ f$› **and** ‹$cont_{\downarrow}\ g$› **and** ‹$cont_{\downarrow}\ \Psi$› **and** ‹$cont_{\downarrow}\ \Omega$›
shows ‹$adm_{\downarrow}\ (\lambda x.\ f\ x \rightsquigarrow_{\checkmark r} (g\ x))$›
proof (*intro restriction-adm-conj*)
show ‹$adm_{\downarrow}\ (\lambda x.\ \checkmark(r) \in (f\ x)^0)$›
by (*simp add*: ‹$cont_{\downarrow}\ f$› *restriction-adm-in-initials*)
next
show ‹$adm_{\downarrow}\ (\lambda x.\ f\ x\ after_{\checkmark}\ \checkmark(r) \rightsquigarrow_{\tau} g\ x)$›
proof (*rule* τ*-trans-restriction-adm*[*OF* - ‹$cont_{\downarrow}\ g$›],
rule restriction-cont-comp[*OF* - ‹$cont_{\downarrow}\ f$›])
show ‹$cont_{\downarrow}\ (\lambda x.\ x\ after_{\checkmark}\ \checkmark(r))$›
by (*simp add*: ‹$cont_{\downarrow}\ \Psi$› ‹$cont_{\downarrow}\ \Omega$› *restriction-cont-After*$_{tick}$)
qed
qed

lemma (**in** *OpSemTransitions*) *restriction-adm-weak-trace-trans* [*restriction-adm-process*$_{ptick}$*-simpset*]:
fixes $f\ g ::$ ‹$'b ::$ *restriction* $\Rightarrow ('a, 'r)\ process_{ptick}$›
assumes τ*-trans-restriction-adm*:
‹$\bigwedge f\ g :: 'b \Rightarrow ('a, 'r)\ process_{ptick}.\ cont_{\downarrow}\ f \Longrightarrow cont_{\downarrow}\ g \Longrightarrow adm_{\downarrow}$
$(\lambda x.\ f\ x \rightsquigarrow_{\tau} g\ x)$›
and ‹$cont_{\downarrow}\ f$› **and** ‹$cont_{\downarrow}\ g$› **and** ‹$cont_{\downarrow}\ \Psi$› **and** ‹$cont_{\downarrow}\ \Omega$›
shows ‹$adm_{\downarrow}\ (\lambda x.\ f\ x \rightsquigarrow^{*} t\ (g\ x))$›
proof (*subst trace-trans-iff-T-and-After*$_{trace}$ τ*-trans*, *intro restriction-adm-conj*)
show ‹$adm_{\downarrow}\ (\lambda x.\ t \in \mathcal{T}\ (f\ x))$› **by** (*simp add*: ‹$cont_{\downarrow}\ f$› *restriction-adm-in-*$\mathcal{T}$)
next
show ‹$adm_{\downarrow}\ (\lambda x.\ f\ x\ after_{\mathcal{T}}\ t \rightsquigarrow_{\tau} g\ x)$›
proof (*rule* τ*-trans-restriction-adm*[*OF* - ‹$cont_{\downarrow}\ g$›])
show ‹$cont_{\downarrow}\ (\lambda x.\ f\ x\ after_{\mathcal{T}}\ t)$›

by (*simp add*: ‹$cont_{\downarrow}\ f$› ‹$cont_{\downarrow}\ \Psi$› ‹$cont_{\downarrow}\ \Omega$› $restriction\text{-}cont\text{-}After_{trace}$)
qed
qed

declare $restriction\text{-}adm\text{-}process_{ptick}\text{-}simpset$ [*simp*]

12 Higher-Order Rules

This is the main entry point. We configure the simplifier below.

named-theorems $restriction\text{-}shift\text{-}process_{ptick}\text{-}simpset$

12.1 Prefixes

lemma $Mprefix\text{-}restriction\text{-}shift\text{-}process_{ptick}$ [$restriction\text{-}shift\text{-}process_{ptick}\text{-}simpset$]
:
‹*constructive* ($\lambda x.\ \Box a \in A \rightarrow f\ a\ x$)› **if** ‹($\bigwedge a.\ a \in A \Longrightarrow$ *non-destructive* (*f a*))›
proof −
have ∗ : ‹$\Box a \in A \rightarrow f\ a\ x = \Box a \in A \rightarrow$ (*if a* ∈ *A then f a x else STOP*)› **for** *x*
by (*auto intro*: *mono-Mprefix-eq*)
show ‹*constructive* ($\lambda x.\ \Box a \in A \rightarrow f\ a\ x$)›
by (*subst* ∗, *rule constructive-comp-non-destructive*
[*OF Mprefix-constructive*, *of* ‹$\lambda x\ a.$ *if a* ∈ *A then f a x else STOP*›])
(*auto intro*: *that*)
qed

lemma $Mndetprefix\text{-}restriction\text{-}shift\text{-}process_{ptick}$ [$restriction\text{-}shift\text{-}process_{ptick}\text{-}simpset$]
:
‹*constructive* ($\lambda x.\ \sqcap a \in A \rightarrow f\ a\ x$)› **if** ‹($\bigwedge a.\ a \in A \Longrightarrow$ *non-destructive* (*f a*))›
proof −
have ∗ : ‹$\sqcap a \in A \rightarrow f\ a\ x = \sqcap a \in A \rightarrow$ (*if a* ∈ *A then f a x else STOP*)› **for** *x*
by (*auto intro*: *mono-Mndetprefix-eq*)
show ‹*constructive* ($\lambda x.\ \sqcap a \in A \rightarrow f\ a\ x$)›
by (*subst* ∗, *rule constructive-comp-non-destructive*
[*OF Mndetprefix-constructive*, *of* ‹$\lambda x\ a.$ *if a* ∈ *A then f a x else STOP*›])
(*auto intro*: *that*)
qed

corollary *write0-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹*non-destructive f* ⟹ *constructive* (λ*x*. *a* → *f x*)›
by (*simp add*: *write0-def Mprefix-restriction-shift-process*$_{ptick}$)

corollary *write-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹*non-destructive f* ⟹ *constructive* (λ*x*. *c***!***a* → *f x*)›
by (*simp add*: *write-def Mprefix-restriction-shift-process*$_{ptick}$)

corollary *read-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹(⋀*a*. *a* ∈ *A* ⟹ *non-destructive* (*f a*)) ⟹ *constructive* (λ*x*. *c***?***a* ∈ *A* → *f a x*)›
by (*simp add*: *read-def Mprefix-restriction-shift-process*$_{ptick}$ *inv-into-into*)

corollary *ndet-write-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹(⋀*a*. *a* ∈ *A* ⟹ *non-destructive* (*f a*)) ⟹ *constructive* (λ*x*. *c***!!***a* ∈ *A* → *f a x*)›
by (*simp add*: *ndet-write-def Mndetprefix-restriction-shift-process*$_{ptick}$ *inv-into-into*)

12.2 Choices

lemma *GlobalNdet-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹(⋀*a*. *a* ∈ *A* ⟹ *non-destructive* (*f a*)) ⟹ *non-destructive* (λ*x*. ⊓*a* ∈ *A*. *f a x*)›
‹(⋀*a*. *a* ∈ *A* ⟹ *constructive* (*f a*)) ⟹ *constructive* (λ*x*. ⊓*a* ∈ *A*. *f a x*)›
proof −
have ∗ : ‹⊓*a* ∈ *A*. *f a x* = ⊓*a* ∈ *A*. (*if a* ∈ *A then f a x else STOP*)›
for *x*
by (*auto intro*: *mono-GlobalNdet-eq*)

show ‹(⋀*a*. *a* ∈ *A* ⟹ *non-destructive* (*f a*)) ⟹ *non-destructive* (λ*x*. ⊓*a* ∈ *A*. *f a x*)›
by (*subst* ∗, *rule non-destructive-comp-non-destructive*
[*OF GlobalNdet-non-destructive*, *of* ‹λ*x a*. *if a* ∈ *A then f a x else STOP*›]) *auto*

show ‹(⋀*a*. *a* ∈ *A* ⟹ *constructive* (*f a*)) ⟹ *constructive* (λ*x*. ⊓*a* ∈ *A*. *f a x*)›
by (*subst* ∗, *rule non-destructive-comp-constructive*
[*OF GlobalNdet-non-destructive*, *of* ‹λ*x a*. *if a* ∈ *A then f a x else STOP*›]) *auto*
qed

lemma *GlobalDet-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹$(\bigwedge a.\ a \in A \Longrightarrow$ *non-destructive* $(f\ a)) \Longrightarrow$ *non-destructive* $(\lambda x.\ \Box a \in A.\ f\ a\ x)$›
‹$(\bigwedge a.\ a \in A \Longrightarrow$ *constructive* $(f\ a)) \Longrightarrow$ *constructive* $(\lambda x.\ \Box a \in A.\ f\ a\ x)$›
proof −
have $*$: ‹$\Box a \in A.\ f\ a\ x = \Box a \in A.$ (*if* $a \in A$ *then* $f\ a\ x$ *else* $STOP$)›
for x
by (*auto intro*: *mono-GlobalDet-eq*)

show ‹$(\bigwedge a.\ a \in A \Longrightarrow$ *non-destructive* $(f\ a)) \Longrightarrow$ *non-destructive* $(\lambda x.\ \Box a \in A.\ f\ a\ x)$›
by (*subst* $*$, *rule non-destructive-comp-non-destructive*
[*OF GlobalDet-non-destructive*, *of* ‹$\lambda x\ a.$ *if* $a \in A$ *then* $f\ a\ x$ *else* $STOP$›]) *auto*

show ‹$(\bigwedge a.\ a \in A \Longrightarrow$ *constructive* $(f\ a)) \Longrightarrow$ *constructive* $(\lambda x.\ \Box a \in A.\ f\ a\ x)$›
by (*subst* $*$, *rule non-destructive-comp-constructive*
[*OF GlobalDet-non-destructive*, *of* ‹$\lambda x\ a.$ *if* $a \in A$ *then* $f\ a\ x$ *else* $STOP$›]) *auto*
qed

lemma *Ndet-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹*non-destructive* $f \Longrightarrow$ *non-destructive* $g \Longrightarrow$ *non-destructive* $(\lambda x.\ f\ x \sqcap g\ x)$›
‹*constructive* $f \Longrightarrow$ *constructive* $g \Longrightarrow$ *constructive* $(\lambda x.\ f\ x \sqcap g\ x)$›
by (*auto intro*!: *non-destructiveI constructiveI*
simp add: *restriction-process*$_{ptick}$*-Ndet dest*!: *non-destructiveD constructiveD*)

lemma *Det-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹*non-destructive* $f \Longrightarrow$ *non-destructive* $g \Longrightarrow$ *non-destructive* $(\lambda x.\ f\ x \mathbin{\Box} g\ x)$›
‹*constructive* $f \Longrightarrow$ *constructive* $g \Longrightarrow$ *constructive* $(\lambda x.\ f\ x \mathbin{\Box} g\ x)$›
by (*auto intro*!: *non-destructiveI constructiveI*
simp add: *restriction-process*$_{ptick}$*-Det dest*!: *non-destructiveD constructiveD*)

lemma *Sliding-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹*non-destructive* $f \Longrightarrow$ *non-destructive* $g \Longrightarrow$ *non-destructive* $(\lambda x.\ f\ x \rhd g\ x)$›
‹*constructive* $f \Longrightarrow$ *constructive* $g \Longrightarrow$ *constructive* $(\lambda x.\ f\ x \rhd g\ x)$›
by (*auto intro*!: *non-destructiveI constructiveI*

simp add: *restriction-process*$_{ptick}$*-Sliding dest*!: *non-destructiveD constructiveD*)

12.3 Renaming

lemma *Renaming-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹*non-destructive P* ⟹ *non-destructive* (λ*x*. *Renaming* (*P x*) *f g*)›
‹*constructive P* ⟹ *constructive* (λ*x*. *Renaming* (*P x*) *f g*)›
by (*auto intro*!: *non-destructiveI constructiveI*
simp add: *restriction-process*$_{ptick}$*-Renaming dest*!: *non-destructiveD constructiveD*)

12.4 Sequential Composition

lemma *Seq-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹*non-destructive f* ⟹ *non-destructive g* ⟹ *non-destructive* (λ*x*. *f x* ; *g x*)›
‹*constructive f* ⟹ *constructive g* ⟹ *constructive* (λ*x*. *f x* ; *g x*)›
by (*fact non-destructive-comp-non-destructive*[*OF Seq-non-destructive non-destructive-prod-codomain*, *simplified*])
(*fact non-destructive-comp-constructive*[*OF Seq-non-destructive constructive-prod-codomain*, *simplified*])

lemma *MultiSeq-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹(⋀*l*. *l* ∈ *set L* ⟹ *non-destructive* (*f l*)) ⟹ *non-destructive* (λ*x*. *SEQ l* ∈@ *L*. *f l x*)›
‹(⋀*l*. *l* ∈ *set L* ⟹ *constructive* (*f l*)) ⟹ *constructive* (λ*x*. *SEQ l* ∈@ *L*. *f l x*)›
by (*induct L rule*: *rev-induct*; *simp add*: *Seq-restriction-shift-process*$_{ptick}$)+

corollary *MultiSeq-non-destructive* : ‹*non-destructive* (λ*P*. *SEQ l* ∈@ *L*. *P l*)›
by (*simp add*: *MultiSeq-restriction-shift-process*$_{ptick}$(*1*)[*of L* ‹λ*l x*. *x l*›])

12.5 Synchronization Product

lemma *Sync-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹*non-destructive f* ⟹ *non-destructive g* ⟹ *non-destructive* (λ*x*. *f x* ⟦*S*⟧ *g x*)›
‹*constructive f* ⟹ *constructive g* ⟹ *constructive* (λ*x*. *f x* ⟦*S*⟧ *g x*)›
by (*fact non-destructive-comp-non-destructive*[*OF Sync-non-destructive non-destructive-prod-codomain*, *simplified*])

(*fact non-destructive-comp-constructive*[*OF Sync-non-destructive constructive-prod-codomain*, *simplified*])

lemma *MultiSync-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹(⋀*m*. *m* ∈# *M* ⟹ *non-destructive* (*f m*)) ⟹ *non-destructive* (λ*x*. ⟦*S*⟧ *m* ∈# *M*. *f m x*)›
‹(⋀*m*. *m* ∈# *M* ⟹ *constructive* (*f m*)) ⟹ *constructive* (λ*x*. ⟦*S*⟧ *m* ∈# *M*. *f m x*)›
by (*induct M rule*: *induct-subset-mset-empty-single*;
simp add: *Sync-restriction-shift-process*$_{ptick}$)+

corollary *MultiSync-non-destructive* : ‹*non-destructive* (λ*P*. ⟦*S*⟧ *m* ∈# *M*. *P m*)›
by (*rule MultiSync-restriction-shift-process*$_{ptick}$(*1*)[*of M* ‹λ*m x*. *x m*›]) *simp*

12.6 Throw

lemma *Throw-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹*non-destructive f* ⟹ (⋀*a*. *a* ∈ *A* ⟹ *non-destructive* (*g a*)) ⟹ *non-destructive* (λ*x*. *f x* Θ *a* ∈ *A*. *g a x*)›
‹*constructive f* ⟹ (⋀*a*. *a* ∈ *A* ⟹ *constructive* (*g a*)) ⟹ *constructive* (λ*x*. *f x* Θ *a* ∈ *A*. *g a x*)›
proof −
have ∗ : ‹*f x* Θ *a* ∈ *A*. *g a x* = *f x* Θ *a* ∈ *A*. (*if a* ∈ *A then g a x else STOP*)› **for** *x*
by (*auto intro*: *mono-Throw-eq*)

show ‹*non-destructive f* ⟹ (⋀*a*. *a* ∈ *A* ⟹ *non-destructive* (*g a*)) ⟹ *non-destructive* (λ*x*. *f x* Θ *a* ∈ *A*. *g a x*)›
by (*subst* ∗, *erule non-destructive-comp-non-destructive*
[*OF Throw-non-destructive non-destructive-prod-codomain*, *simplified*]) *auto*

show ‹*constructive f* ⟹ (⋀*a*. *a* ∈ *A* ⟹ *constructive* (*g a*)) ⟹ *constructive* (λ*x*. *f x* Θ *a* ∈ *A*. *g a x*)›
by (*subst* ∗, *erule non-destructive-comp-constructive*
[*OF Throw-non-destructive constructive-prod-codomain*, *simplified*]) *auto*
qed

12.7 Interrupt

lemma *Interrupt-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:

‹*non-destructive f* ⟹ *non-destructive g* ⟹ *non-destructive* (λ*x*. *f x* △ *g x*)›
‹*constructive f* ⟹ *constructive g* ⟹ *constructive* (λ*x*. *f x* △ *g x*)›
by (*fact non-destructive-comp-non-destructive*[*OF Interrupt-non-destructive non-destructive-prod-codomain*, *simplified*])
(*fact non-destructive-comp-constructive*[*OF Interrupt-non-destructive constructive-prod-codomain*, *simplified*])

12.8 After

lemma (**in** *After*) *After-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹*non-too-destructive* Ψ ⟹ *constructive f* ⟹ *non-destructive* (λ*x*. *f x after a*)›
‹*non-too-destructive* Ψ ⟹ *non-destructive f* ⟹ *non-too-destructive* (λ*x*. *f x after a*)›
by (*auto intro*!: *non-too-destructive-comp-constructive*[*OF non-too-destructive-After*]
non-too-destructive-comp-non-destructive[*OF non-too-destructive-After*])

lemma (**in** *AfterExt*) *After*$_{tick}$*-restriction-shift-process*$_{ptick}$ [*restriction-shift-process*$_{ptick}$*-simpset*]
:
‹⟦*non-too-destructive* Ψ; *non-too-destructive* Ω; *constructive f*⟧
⟹ *non-destructive* (λ*x*. *f x after*$_{✓}$ *e*)›
‹⟦*non-too-destructive* Ψ; *non-too-destructive* Ω; *non-destructive f*⟧
⟹ *non-too-destructive* (λ*x*. *f x after*$_{✓}$ *e*)›
by (*auto intro*!: *non-too-destructive-comp-constructive*[*OF non-too-destructive-After*$_{tick}$]
non-too-destructive-comp-non-destructive[*OF non-too-destructive-After*$_{tick}$]
simp add: *non-too-destructive-fun-iff*)

12.9 Illustration

declare *restriction-shift-process*$_{ptick}$*-simpset* [*simp*]

notepad begin
fix *e f g* :: *'a* **fix** *r s* :: *'r*
fix *A B C* :: ‹*'a set*›
fix *S* :: ‹*'a* ⇒ *'a* ⇒ *'a* ⇒ *'a set*›
define *P* **where** ‹*P* ≡ υ *X*. ((□*a* ∈ *A* → *X* ⊓ *SKIP r*) △ (*f* → *STOP*))
□ (*g* → *X*)
□ ((*f* → *e* → (⊥ ⊓ (*e* → *X*))) Θ *b* ∈ *insert e B*. (*e* → *SKIP s*))›
(**is** ‹*P* ≡ υ *X*. *?f X*›)
have ‹*constructive ?f*› **by** *simp*
have ‹*cont ?f*› **by** *simp*
have ‹*P* = *?f P*›
unfolding *P-def* **by** (*subst restriction-fix-eq*) *simp-all*

define Q **where** ‹$Q \equiv \upsilon\ X.\ (\lambda\sigma\ \sigma'\ \sigma''\ .\ e \to \sqcap\ b \in S\ \sigma\ \sigma'\ \sigma''.\ X\ b\ b\ b \sqcap\ SKIP\ r)$› (**is** ‹$Q \equiv \upsilon\ X.\ ?g\ X$›)
have ‹*constructive ?g*› **by** *simp*

define R **where** ‹$R \equiv \upsilon\ (x,\ y).\ (e \to y \sqcap\ SKIP\ r,\ \Box a \in A \to x)$›
(**is** ‹$R \equiv \upsilon\ (x,\ y).\ (?h\ y,\ ?i\ x)$›)
have ‹*snd* $R = \Box a \in A \to$ *fst* R›
by (*unfold R-def*, *subst restriction-fix-eq*)
(*simp-all add*: *case-prod-beta′*)

end

end